DDI-Cross Domain Integration: Detailed Model

Contents

I. The Upper Model

The purpose of this section is to show how DDI – Cross Domain Integration (DDI – CDI) fits into the overall context of research and the metadata which is used to describe research activities and resources. The upper model touches on some areas which are not covered in detail by other parts of DDI - CDI, but which may be important in terms of understanding how it can be implemented in systems or with other standards that do cover those areas. The Upper Model is merely diagrammatic: it is not included as part of the detailed DDI - CDI Model and is not found in the syntax representations. It is intended to provide context for how the model may be used in relation to other real-world entities within systems.

Note that an "upper model" (also known as an "upper ontology" and "foundation ontology") functions "to support broad semantic interoperability among a large number of domain-specific ontologies by providing a common starting point for the formulation of definitions. Terms in the domain ontology are ranked under the terms in the upper ontology, e.g., the upper ontology classes are superclasses or supersets of all the classes in the domain ontologies" (Wikipedia).

A Research Program for the purposes of this model is an undertaking intended to produce and/or analyze data which measures real-world phenomena for the purpose of answering questions about policy and science, understood in its broadest sense. It is the context for data capture, data management, data analysis and /or data dissemination, depending on the actual scope of the Research Program.

The diagram below shows the Research Program together with a Research Methodology that guides it:

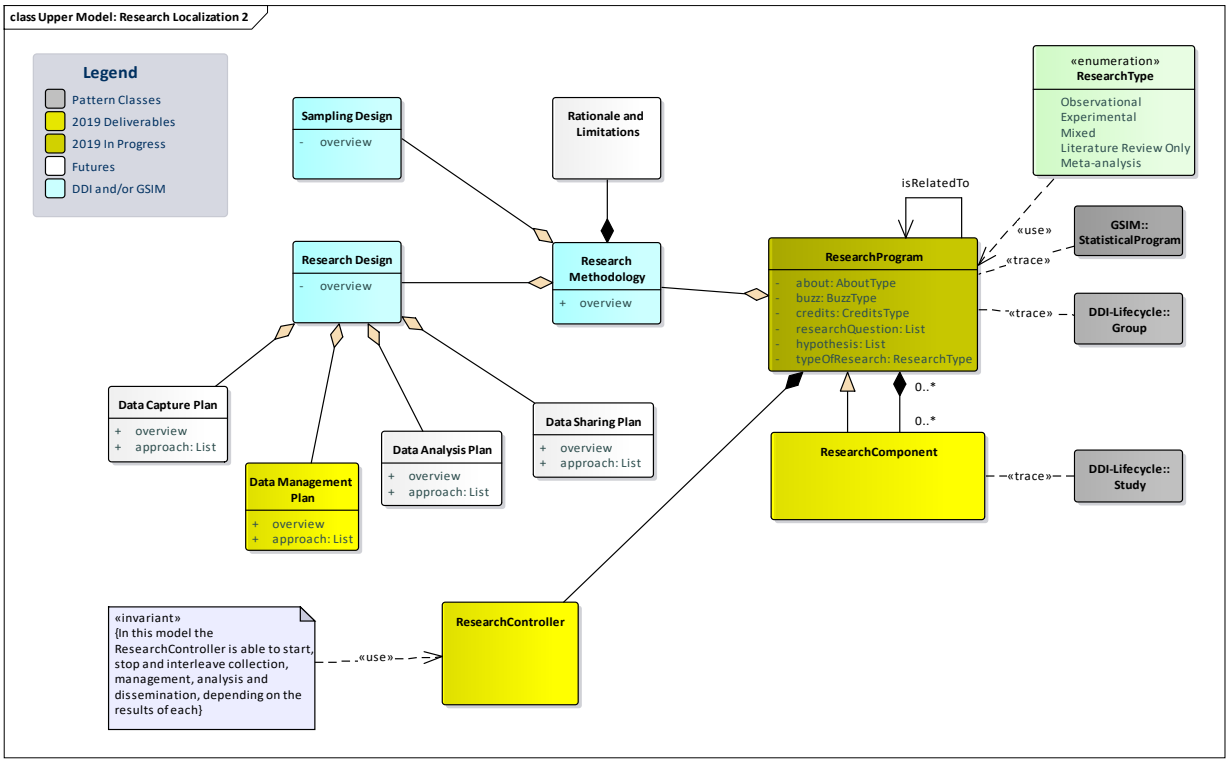


Figure 1: Research Program and Research Methodology

In this model the Research Program corresponds to a GSIM Statistical Program and the Group from DDI Lifecycle. Here a Research Component might be a "round" in a longitudinal study. However, a Research Program need not be a longitudinal study. Instead of a time series, it could also be the umbrella for a group of more or less related snapshots -- a research "mosaic", if you will -- that an organization undertakes, depending upon what the Research Program and its Research Components are "about".

The "about" property group of a Research Program and its Research Components is a structured set of data based on terms from DDI Codebook, Dublin Core and Schema.org, assuring comparability across many standards. Comparability is assured both within the DDI family of standards as well as between DDI and other standards. Note that the "about" of a Research Program and its Research Components includes "coverage" and "provenance" at various levels of specificity in line with Dublin Core, DDI Lifecycle, and Schema.org.

A Research Program also has "buzz" and "credits" property groups. "Buzz" comes from Schema.org and is the profile that a Research Program cuts in social media – its interaction statistics, audience characteristics, comment characteristics, trolls and similar aspects. Between "about", "buzz" and "credits" the Research Program supports the construction of many types of bibliographic citations. (To understand how these property groups were developed, see the document "DDI Cross Domain Integration: Architecture and Alignment with Other Standards" in this review package.)

A Research Program is informed by and conducted according to a Research Methodology, which involves the plan for how it will be conducted, represented by the Research Design class. Research Methodology here is generic. It is patterned after the Research Methodology section or chapter in a thesis or a research paper in line with the recommendations of many scientific and governmental organizations. It is not intended to be machine actionable. Indeed, DDI - CDI does not provide a detailed model of methodology or data management planning. Instead, it focuses on the data and processes which such methodologies and plans might involve. One exception to this overall approach is that Sampling Design is broken out. Sampling Design is a placeholder for a sampling plan model able to describe sampling plans for surveys and other statistical activities for all kinds of samples including probability, non-probability and multi-stage / multi-frame.

A Research Program is controlled by a Research Controller – in PROV-O terms an Agent. The Research Controller starts, stops and interleaves data collection, data management, data analysis and data dissemination, depending on the results of each, in line with the Research Methodology.

The diagram below shows in more detail how the Research Controller relates to other points of focus within DDI - CDI, notably process and data description.

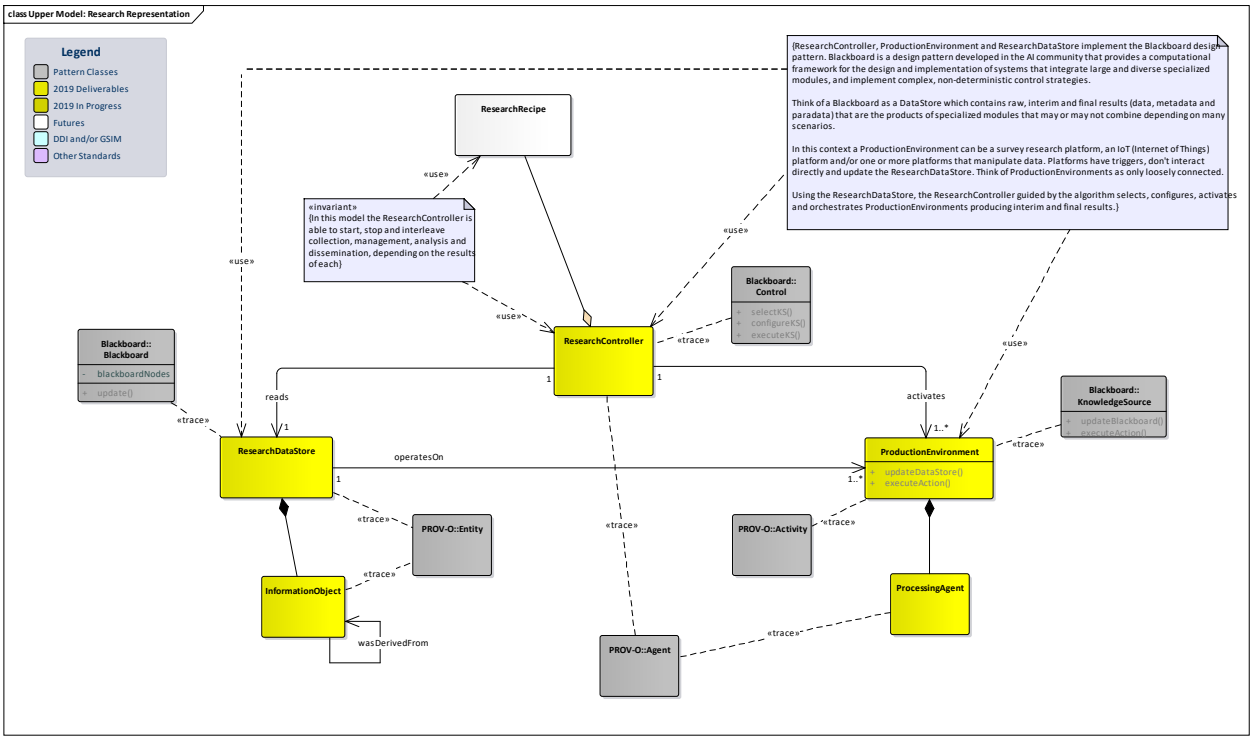


Figure 2: The Research Controller

The Research Controller interacts with two important parts of the model -- the Research Data Store and the Production Environment. The Research Data Store may be a repository, a virtualized data store or a metadata store that represents the data at its many stages of development which are used by and produced during the Research Program. DDI - CDI provides a model for describing many of the types of data which Research Data Stores contain. This model is extensive and supports the integration and transformation of the different types of data required.

The Research Controller also interacts with the Production Environment. DDI - CDI includes two types of production environments for now – the Data Capture Platform and the Data Processing Platform.

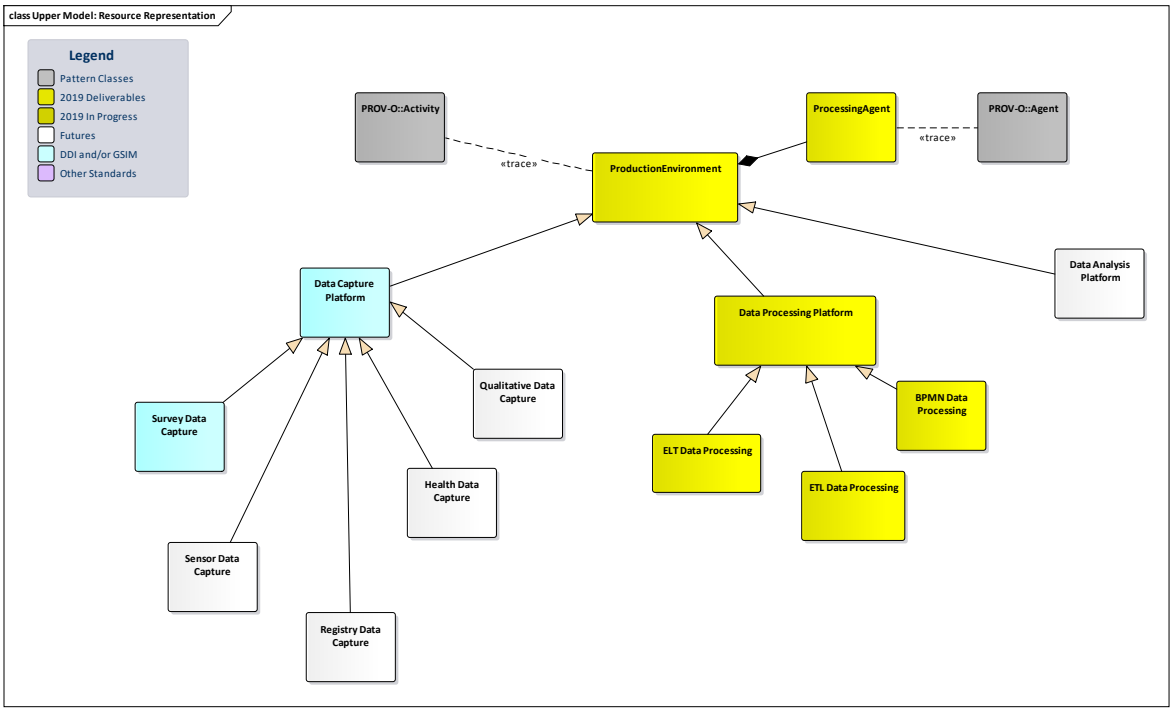
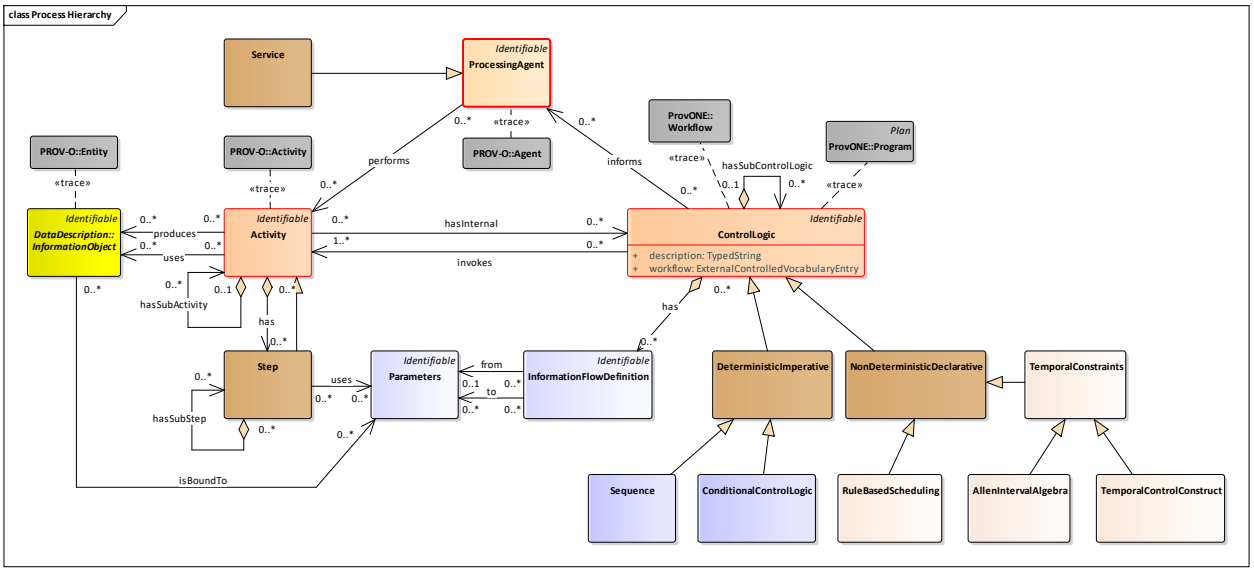


Figure 3: Production Environments

The Data Capture Platform includes Survey Data Capture together with other data capture types. Several types of Data Processing Platforms are depicted here including the ETL (Extract, Transform, and Load) , ELT (Extract, Load, and Transform), and BPMN Platforms. These types and the several data capture platform types shown here are placeholders for models that may have been developed by other standards. Treatment of these placeholders by the production system and at instantiation time by tools developed in various communities of practice is discussed in the Architecture document sections on standards alignment.

What DDI - CDI, however, does include is a detailed Process Model. What is presented here is a very high-level description. The detailed description can be found in Section IV of this document.



F*igure 4: Process Model High Level*

The DDI – CDI Process Model is intended to be usable in every type of Production Environment. It supports the description of both deterministic and non-deterministic control systems. Deterministic control systems use deterministic control logic. Non-deterministic control systems include rules-based systems and systems that employ machine learning.

Note that the DDI Core Process Model traces to PROV-O and an extension of PROV-O called ProvONE. This is because the process descriptions in the DDI - CDI model are intended to support the description of process as provenance documentation. This is significant – in past versions of DDI, the process model has been designed to both describe workflows (DDI 4 Prototype) and the flow of questionnaires and related processing (DDI Lifecycle, DDI 4 Prototype) in sufficient detail that they can drive the execution of data processing workflows and questionnaires. In DDI - CDI the focus of the process description is restricted to just the documentation of provenance. Note that DDI Core, like ProvONE, extends the "entity" in PROV-O to data at the variable level, the documentation of specific workflow types and the evolution of workflows as occurs in machine learning.

The Research Data Store comprises a set of Information Objects of different types. Significant among these are data sets. DDI - CDI provides a detailed description of data and the structures which are used by different types of data sets.

Some of the most significant Information Objects – those related to describing data – are a major focus if the DDI - CDI model. The diagram below shows a more detailed view of how data fits into the Production System.

Note that DDI - CDI Information Objects relate to classes in some other popular models: PROV-O provides us with the Entity, to which Information Objects correspond; they can also be seen as equivalent to GSIM Information Resources.

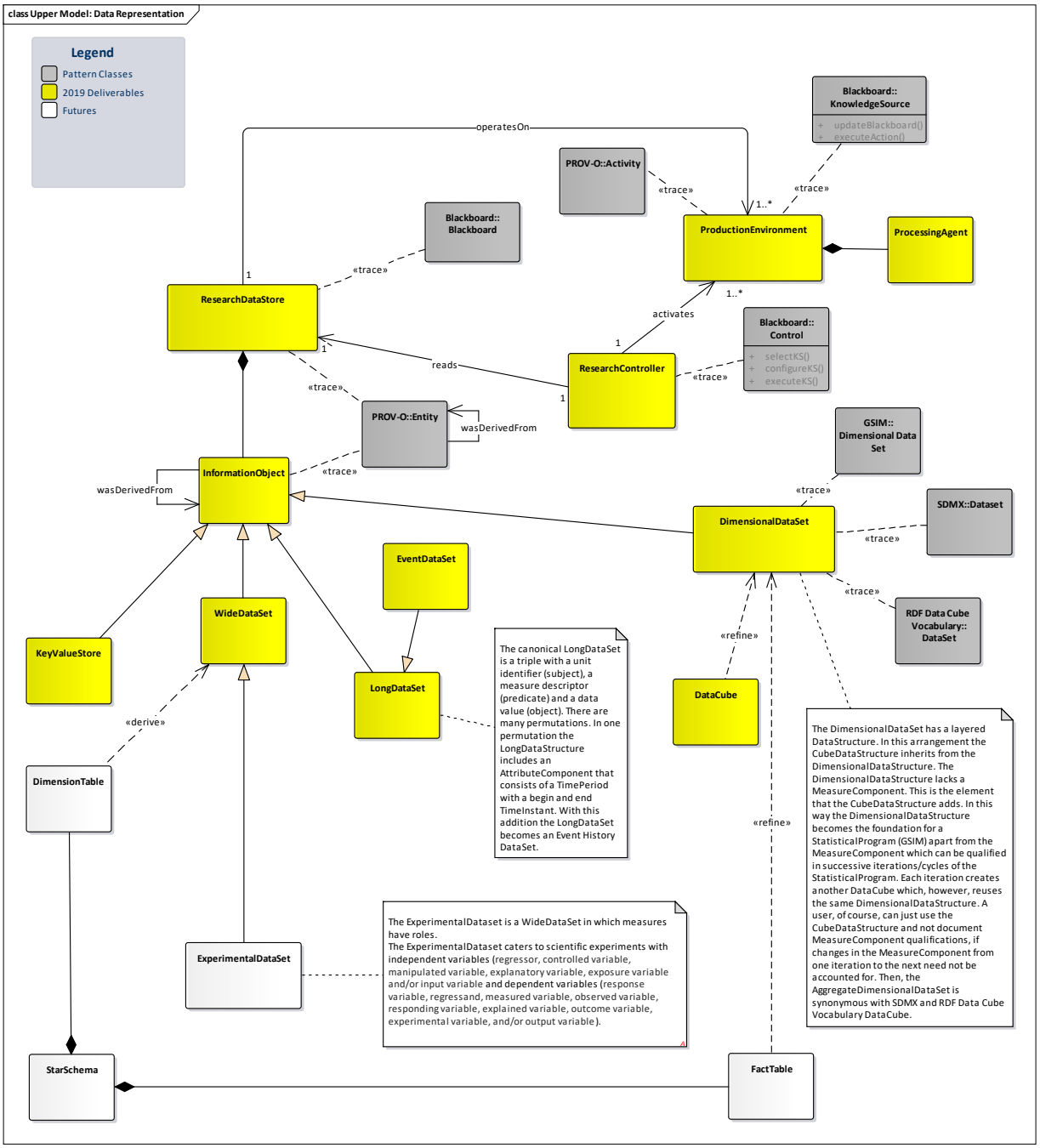


Figure 5: Data Representation

The mechanism for describing data structures in the DDI - CDI model provides four basic types: Wide Data, Long Data, Multi-Dimensional Data, and Key-Value Data. These types represent different styles of describing data structures, using a consistent set of components for identification, grouping observations into records, adding descriptive fields, and so on. The differences between each type of data description are found in the roles played by these different components.

The four types are characterized as:

Wide Data: This is a way of describing traditional rectangular unit-record data sets. Each record has a set of observations about a single unit. The record has a unit identifier and a set of measures and/or descriptors which are the same for each unit. The unit identifier can be used as an identifier for the record, because each unit has only one.

Long Data: This is a technique for describing many common types of data, including sensor data, event data, and spell data. In this form, each record has a unit identifier and a set of measures and/or descriptors, but there may be multiple records for any given unit. The identification of the record is a combination of the unit identifier and one or more other fields.

There are two refinements of the Long Data class which further constrain it to correspond with Event Data (defined as Long Data for which an observation time is provided as a single point) , and Spell Data (Long Data with fields for start and end times bounding a period). In both of these, the record identification involves time as well as the unit identifier.

Multidimensional Data: Multi-dimensional data is data in which observations are identified using a set of dimensions. These values both identify the cell and serve to describe the measured population. Records may be organized in various ways and may include descriptors as well as their dimensions and measures. It is common to view such data sets as multi-dimensional Cubes, and also to describe them as Time Series. These specific approaches are defined as sub-types of DDI - CDI's multi-dimensional data.

Key-Value Data: Key-Value Data is data which consists of a set of measures, each of which is paired with an identifier. Descriptors may also be attached to these pairs. Such data is often organized in more complex ways when it is used but may be stored or exchanged using this simple construction.

II. Foundational Metadata: Concept, Datum, and Variable

A. Introduction

This section explains concepts, data, and variables as used and described in DDI - CDI. It is detailed and technical, but the language and ideas are accessible. They are based on everyday experience; however, the approach may be unfamiliar. The result provides a thorough understanding of concepts, data, and variables as used in DDI - CDI. The approach uses the theory of terminology as described in ISO 704 – *Terminology – Principles and methods*.

The underlying theory for understanding data and variables is the same as that for concepts and terms. We develop this connection. We start with a full description of concepts, what they are, how they are structured, and how people refer to them. From this we show that data are a kind of terminology, and the connection between concepts and their corresponding objects is precisely what variables do.

Data are often described by what they do, the operations and statistics available to process them. The terminological approach is an attempt to define what data are.

Variables are described in DDI - CDI through levels of specificity. This is known as the variable cascade, and it enhances reuse of metadata, an important principle of metadata management. How the cascade ties back into the terminological view of data and variables is described.

B. Theory of Terminology

The theory of terminology for special language is described in ISO 704 – *Terminology – Principles and methods*. This section is a reformulation of that standard to data and variables. The ideas of concept and object are used throughout.

Readers might find this overall section very philosophical. It is not intended to be that way. We adopt a mentalist position for concepts and nothing more. This corresponds to experience. Likewise, objects are very generally defined, and they correspond to things in the world that people and systems use. We hope this approach allows the reader to maintain an intuitive understanding.

1. Objects

An object is anything perceivable or conceivable. This is a very general definition, and it implies that the idea of object, for our purposes, is the most general thing we consider. Each thing, physical or not, is an object.

Perceivable objects are those detectable through one of the five human senses. Any physical object in the world is perceivable, mostly through sight and touch, but the other senses may be used as well. For instance, a sound is perceivable through hearing. An object may also be perceived through some detector. Examples include voltage and current (in electricity), and they are perceived through instruments.

Objects may also be conceivable, and these come in two main kinds: abstract and imaginary. Examples of abstract conceivable objects are variables, laws, and numbers. Examples of imaginary conceivable objects are unicorns and hallucinations.

2. Properties and Characteristics

A **property**[1] is a determinant, the result of a determination either directly or indirectly about some object. Note, this term is used in many other subject areas, and its meaning here is close to the others. However, there is a precise and specific meaning being used here.

One form of determination is through observation – something humans perceive through their senses. Noticing the color of a person's eyes is an observation or direct determination of the eye color of that person. Another form of determination is through detection by an instrument. An oral thermometer is an instrument that detects internal body temperature of a person. Observing a reading on the thermometer is an indirect determination about the internal temperature of a person. The specific observed eye color and internal body temperature are properties of a person.

It is through properties that enable us to describe and make distinctions between objects. For instance, one person may be 185 cm tall, have brown colored eyes and hair, and have medium brown colored skin. Another may be 170 cm tall, have blue colored eyes and blond hair, and have very light brown colored skin. These properties of each person serve to help distinguish between the two.

[1] The term property is not defined in ISO 704.

Since the examples above use perceivable objects, it is important to note that conceivable objects have properties, too. For instance, consider the rational numbers “three and fourteen hundredths” and “negative seventeen”. In the same way as with perceivable objects, properties of conceivable objects are the results of determinations about these objects. Here, the “sign” (in the mathematical sense) of the numbers is a property of them. The “sign” of 3.14 is positive, and the “sign” of -17 is negative.

A **characteristic** is a determinable. A determinable is something capable of being determined, definitely ascertained, or decided upon. It implies a question. Eye color, for instance, is a determinable, and applied to a person begs the question of what the color of the eyes for that person is. It is capable of being ascertained by looking into a person’s eyes to determine their color. A property, on the other hand, is the outcome, or determinant, and it is the answer to the question posed by the determinable. A determinant is an element that determines or identifies the nature of something. Blue is a determinant for eye color. So, a characteristic has the capacity for being determined (determinable), whereas the property is the result of a determination (determinant). Some characteristics of a person are height, eye color, hair color, and skin tone. Examples of corresponding properties, taken from the paragraph above, are: height has the properties 185 cm and 170 cm; eye color has the properties brown and blue; hair color has the properties brown and blond; and skin tone has the properties medium brown and very light brown.

A set of properties corresponds to a characteristic. These properties (those in a set) form an extensional definition (See sub-section on Definitions) of the characteristic they correspond to. In Examples 5 and 6, different sets of properties may correspond to the same characteristic, depending on needs. In addition, the same property may correspond to two characteristics. The following Example 1 illustrates this.

EXAMPLE 1: A property may correspond to two characteristics. Consider the following characteristics: height (of a person) and length of the diagonal (of a television screen). The property 60 inches (5 feet or 152.40 cm) corresponds to both characteristics. Some people are 60 inches tall and some large widescreen television sets measure 60 inches diagonally across the screen.

3. Concepts

A **concept** is a unit of thought differentiated by a set of characteristics. Consider the concept “person”. The characteristics of a person include being designed to stand upright on two legs, ability to talk, age, marital status, and skin tone. There are many others.

Some characteristics are indispensable for understanding a concept. These are the **essential characteristics**. A **delimiting characteristic** is a characteristic used to distinguish it from a generic concept. For example, an essential characteristic of people is they are designed to stand and walk upright. This is also a delimiting characteristic since it distinguishes people from other primates.
The **intension** of a concept is the set of characteristics associated with the concept. The **extension** of a concept is the totality of objects to which a concept corresponds.

A **defining characteristic** is a characteristic which is representative of objects in the extension of a concept. A defining characteristic of people is that they stand and walk upright. Not every person is capable of walking and standing upright, even though they are designed that way. Paralyzed or injured people may not be able to stand.

Characteristics and properties are concepts, and each kind plays a role. The role is how the ideas are distinguished. The role for a characteristic is determinable, and that for a property is determinant.

Example 2 illustrates the importance of establishing essential characteristics for a concept. In particular, the addition of a single characteristic may have profound influences on the objects in the extension of the concept. Adding or removing characteristics often affects the meaning of a given concept, changing the concept itself. Thus, the extension would be expected to change.

EXAMPLE 2:

The concept of planet was revised in 2006 by the International Astronomical Union. This revision resulted in the elimination of Pluto as one of the planets in the solar system. Pluto was long considered the ninth planet in the solar system, but some astronomers questioned this classification. Several properties Pluto possesses differ markedly from those of the other planets. Additionally, recent advances in astronomy - much better telescopes and vastly improved computation - showed there are many more celestial bodies that could be considered planets if Pluto remained one. Therefore, a concerted effort was made to define “planet” in a more limiting way.

The concept of a planet is now defined by these four essential characteristics: A planet is a celestial body that

1 Is in orbit around a star
2 Contains sufficient mass to maintain a nearly spherical shape due to its own gravity
3 Is not massive enough to cause thermonuclear fusion in its core
4 Has “cleared the neighborhood”, i.e., become gravitationally dominant, so the only other bodies in its vicinity are its satellites

This fourth characteristic is what eliminated Pluto.

A **general concept** is a concept which corresponds to an indeterminate number of objects which form a group by reason of common properties. An example is the concept “planets in our solar system”. An **individual concept** is a concept which corresponds to only one object. An example is the concept “Saturn”. In other words, a general concept may have any number of objects in its extension, and an individual concept must have exactly one object in its extension.

Note, a concept might be so defined that there exists only one object in its extension even though the possibility for more exists. This is still a general concept. For example, the notion “all planets with one moon” is a general concept. Even though there is one known planet with one moon – Earth – the possibility there are more cannot be ruled out.

The following Figure 6 shows the relationships between concepts and characteristics on the one hand and objects and properties on the other. The figure illustrates the correspondence between a concept and all the objects in its extension. The parallels between this construction and how data are obtained

through surveys, experiments, clinical settings, and other kinds of observations is clear. This parallel will be discussed in subsequent sections.

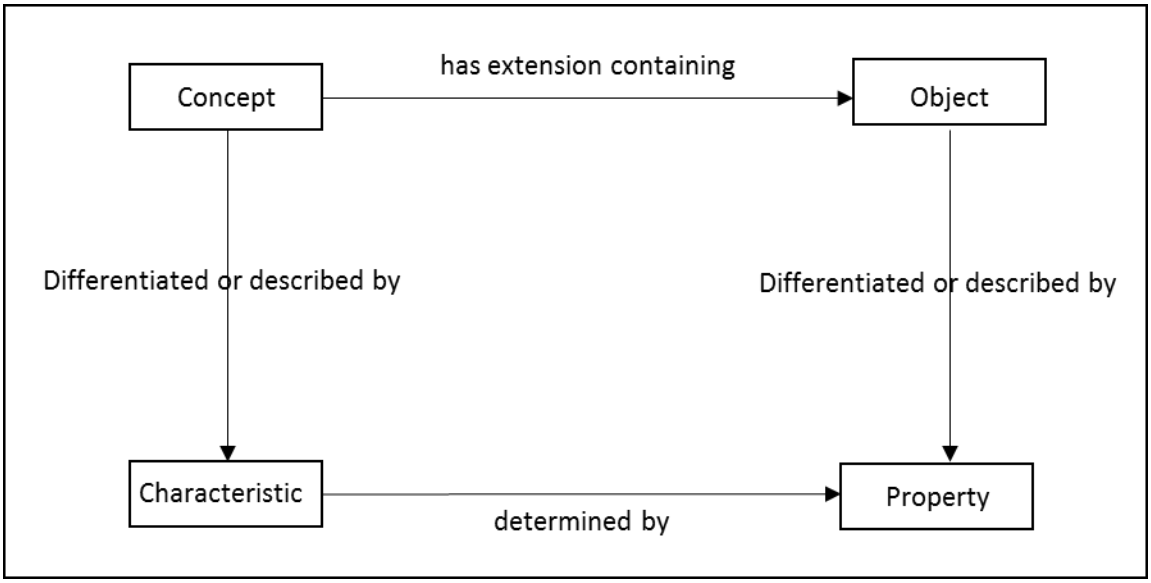


Figure 6: Concept - Object Correspondence

4. Signifier, Signified, and Sign[2]

A **signifier** is a concept whose extension is restricted to perceivable objects. An object in the extension of a signifier is a **token**. For instance, the objects **5** and **5** are both tokens of "the numeral five", a signifier. A signifier has the potential to refer to something. Typically, in statistical offices, the tokens of signifiers are alphanumeric strings.

A **signified** is an object intended to be denoted by a signifier. Any concept, which is also an object, may be a signified. A **sign** is the representation of a signified by a signifier, which denotes it.

See Figure 7 for a pictorial explanation of signs, consistent with the wording in this section.

[2] This is outside the scope of ISO 704.

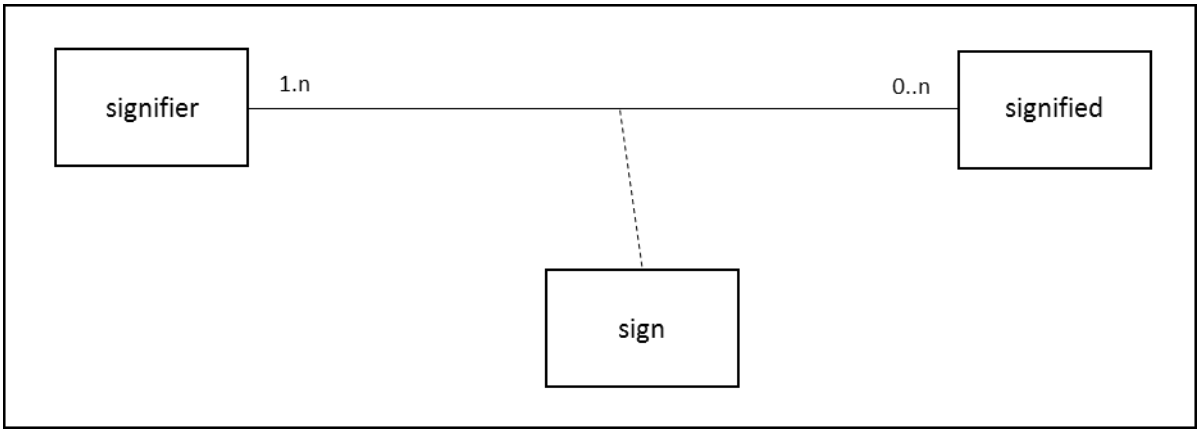


Figure 7: Structure of Signs

5. Kinds of Signs

The following is a list of kinds of signs for which the signified is an object:

- A **label** is a linguistic sign for an object
- A **name** is a non-linguistic sign for an object, where the signifier is an alphanumeric string
- An **identifier** is a label or name intended to be used for dereferencing
- A **locator** is an identifier with a known dereferencing mechanism

The following is a list of kinds of signs for which the signified is a concept:

- A **designation** is a sign for a concept
- A **code** is a non-linguistic designation for which the signifier is alphanumeric
- An **appellation** is a linguistic designation for an individual concept
- A **term** is a linguistic designation for a general concept
- A **numeral** designates a number, where a number is a concept, and the tokens for numerals are numeric strings

Terms, numerals, and codes are typically used to designate values (See section on Data).

6. Definitions

A **definition** is a descriptive statement which serves to convey the meaning for a concept, and it differentiates it from related concepts. There are 2 kinds of definitions. An **intensional definition** is a definition that describes the intension of a concept by stating the superordinate concept and the delimiting characteristics. The definition of delimiting characteristic in the section on Concepts is an example of an intensional definition. An **extensional definition** is a definition of a concept formed by enumerating its subordinate concepts under one criterion of subdivision. The definition of relation in the glossary is an example of an extensional definition. Note, both kinds of definitions depend on knowing the definitions of other concepts in order to fully understand the concept under study.

C. Data

This section contains a description of the connection between data and terminology. A datum is defined as a kind of designation.

1. Values

A **value** is a concept with an equality operation defined.

Any concept may have an equality operation defined for it. For a set of values, the same equality operation is sometimes defined for the entire set, and this leads to the construction of datatypes. See ISO/IEC 11404 – *General purpose datatypes*. Assigning an equality operation to a concept implies that if, say, two people say they have that concept, a determination of equality between them can be made. For example, two people agree they have the same gender. This operation may be different depending on the situation. In fact, more than one measure of equality can be defined for any given concept. See Example 3.

EXAMPLE 3: Consider the natural number "seventeen". It is a concept, and its extension is all situations of 17 objects. Equality may be defined as it is commonly understood for natural numbers. Another way to define equality for natural numbers, including "seventeen", is to ask if the number is even or odd. In this situation, all odd numbers are equal, and all even numbers are equal.

2. Datum

A **datum** is a designation of a value.

A fundamental requirement for a datum is that it can be copied. In Information Technology, the need for copying happens all the time in data processing. The only way to know a datum has been faithfully copied is to compare the copy to the original. The comparison determines whether equality is satisfied (and the copy is faithful). Therefore, a datum must satisfy the equality <11404> property.

EXAMPLE 4: M for married, as in some person is married. Married is a value, since marriage is a social and legal status controlled by the state. Equality may be determined by referencing the meaning in common law.

A datum is often generated in some context, and this context is what connects it to Figure 6 and to the connection between concepts and objects. Suppose we consider the object Donald Trump, and we determine he has orange hair. Donald Trump is an object, and we can find a concept for which he is in its extension. We know, for instance, he is president of the United States, so he is in the extension of the concept of presidents of the US. This concept has characteristics, and one of them is hair color (of a president). For argument's sake, suppose all the possible hair colors of presidents are Orange, Gray, and Brown. Thus, each president (each object in the extension of the concept presidents of the US) has one of the possible hair colors. Washington's hair color was Gray, and Obama's is Brown. In each case, the appropriate one must be determined. So, the possible hair colors are determinants, and they are possible properties of the characteristic hair color.

Now, given that the hair colors Orange, Gray, and Brown are all the ones possible, every president is assigned one and only one color. Assuming the extension of a concept is a set, this means hair color forms a partition of the extension of presidents of the US. Each class in the partition is defined by one of the properties. In this case, there are 3 classes: Orange, Gray, and Brown. No president belongs to more than one class, and every president belongs to at least one. This characterizes a partition.

When we determine the hair color of a president, we might want to record that, so we assign signifiers to each of the possible properties: for instance, o for Orange, g for Gray, and b for Brown, and through this assignment we create designations called codes. Again, by observation, we have a way to decide if two presidents have the same hair color, and this is based on light reflectivity and color reception in the judge's eyes. So, there is an equality operation for each of these properties. This means each of the properties is a value, each code is a designation, and when we assign a hair color and write down a signifier representing the determination, a datum is produced.

Here, the equality operation for each property (value) serves as an equivalence relation for the partition. Two objects (presidents) have the same property and are in the same class in the partition if and only if they have equal hair color. This equality is assessed through the equality operation previously defined.

EXAMPLE 5: Example of a partition of people based on marital status
Concept = people of the UK
Characteristic = marital status
Partition = {single, married, divorced, widowed}, where "single" means never married and the rest correspond to their usual meanings. The signifiers S, M, D, and W designate these concepts, respectively.

EXAMPLE 6: Second example of a partition of people based on marital status
Concept = people of the UK
Characteristic = marital status
Partition = {single, married}, where "single" means not married and married takes its usual meaning. The signs S and M designate these concepts, respectively. The purpose of the example is to show that more than one partition may apply to a characteristic of a concept.

EXAMPLE 7: Example of a partition of gambling casino games based on probability of winning
Concept = gambling casino games
Characteristic = probability of winning
Partition = $\{x \mid 0 < x \leq 1\}$ (the set of all numbers, x, such that x is greater than zero and less than or equal to one), where x is a probability. The signs are the numeric strings that designate the numbers, to some agreed upon precision, fixing the lengths of the strings.

D. Variables and Aggregates

Variables and aggregates are determinable (in the sense previously described), and therefore are characteristics of some concepts. Variables are mostly characteristics for general concepts, and aggregates are mostly characteristics of individual ones. This corresponds to the notion that a variable is a mapping between some collection of units (the extension of the general concept for which the variable is a characteristic) to a set of values. An aggregate does the same, except the concept is an individual one, so there is one unit – the aggregate.

There are some exceptions. In socio-economic statistics, a household income is the sum (an aggregation) of the incomes of each of the individuals in that household. This aggregate applies to a general concept (i.e., households).

Table 1 shows how the terminological constructs described correspond to common notions about data found in socio-economic data.

Socio-Economic Data	**Terminology**
Unit Type or Universe	Concept
Microdata	General concept
Macrodata	Individual concept
Frame	Extension
Variable or aggregate	Characteristic
Unit	Object (in the extension of the concept)
Observation (or estimation)	Property

Table 1: Socio-Economic Data versus Terminology

E. Variable Cascade

In DDI - CDI, the variable cascade is the way the descriptions of variables is managed. The main purpose of the cascade is to increase the reuse of metadata. The features defined at each level of the cascade don't depend on features at any of the lower levels. Because of this, the descriptions at each level are reusable.

The cascade consists of four levels, each level corresponding to an ever-increasing descriptive detail. The levels in the cascade are

- Concept
- Conceptual variable
- Represented variable
- Instance variable

The names of the levels indicate to the user what the main focus of the description is at each. The Concept and Conceptual Variable provide details about the concepts employed. The Represented Variable and Instance Variable provide the details about the codes, characters, and numbers representing the concepts at the higher levels.

We will describe these levels and show how they fit into the terminological approach in the following sections. In tables in each section, we illustrate the approach with two examples. The attributes are taken from the class diagram of DDI - CDI. We only illustrate the attributes at each level. The inherited ones from the level above are assumed.

1. Concept

The variables about some subject share that subject as common among them all. For example, all variables in use in data sets in a research library about marital status share that concept among them all.

There may be little in common about the marital status as measured in each variable, but marital status itself – the fact there are statuses across societies or cultures – is a common characteristic. The concept expressing this commonality is the purpose of this highest level.

The concept at this level is very generic, because it must account for all possible variations of the more specialized versions attached to each variable that makes use of it.

Concept

ID	***Name***	***Definition***
1	Marital status	Category of current marital arrangement
2	Age	Whole number of years of operation

2. Conceptual Variable

The Conceptual Variable is the level at which most of the concepts used to describe a variable are applied. The main concepts are the determinable associated with a variable and the possible determinants. In our marital status example, the main concepts are:

- Determinable: marital status
  - The specialized nature of this concept is that it is applied to people living in the US (for instance)
- Determinants: kinds of marital status
  - Single
  - Married
  - Divorced
  - Widowed

This example illustrates that at the conceptual variable, the determinable and determinants are concepts. Suitable determinants form an extensional definition for the determinable. In our case, single, married, divorced, and widowed do form an extensional definition for marital status. The determinants are known as substantive values in DDI - CDI.

Additional concepts are those associated with missing data. These are known as sentinel values. The two most important, ones that the statistical packages use, are “missing” and “refused”. There might be others, depending on processing needs.

Conceptual Variable (Links to Concept)

Name	Marital Status	Age
Concept	Concept #1	Concept #2
Unit type	Person	Business establishment

Substantive Conceptual Domain	Single Married Divorced Widowed	Count (of years), top coded at 25
Sentinel Conceptual Domain	Missing Refused	Missing Refused

In this table, the names of the categories for marital status in the substantive conceptual domain are there in place of actual concepts. The only way to write down a concept is either through providing a definition or providing an unambiguous term or word denoting it.

3. Represented Variable

The main addition at the Represented Variable level is the signifiers for the determinants, or substantive categories. Assigning signifiers to concepts turns them into designations. So, in our example, we might end up with the following designations:

- <s, single>
- <m, married>
- <d, divorced>
- <w, widowed>

The set of these designations is a substantive value domain. As discussed, the underlying concepts form an extensional definition for the determinable, the concept associated with the variable. So, these determinant concepts are associated with the subject matter of the variable, not with processing. A substantive value domain can be used by many Represented Variables, so it is important to identify and manage them.

Represented Variable (Inherits from Conceptual Variable)

Name	Marital Status	Age
Universe	Deer Hunters	Gun Shops
Substantive Value Domain	<s, Single> <m, Married> <d, Divorced> <w, Widowed>	Count (of years) represented with 2-digit Arabic numeral
Unit of Measure	N/A	years

Intended Datatype family	Nominal	Quantitative discrete

The Intended Datatype Family attribute above needs some explanation. The interpretation of datatypes in this document is contained in the international standard *ISO/IEC 11404 – General purpose datatypes.* This standard provides 2 ideas that are central to understanding the Intended Datatype Family attribute here.

First, each datatype is defined through 3 ideas:

1. ***Value Space*** – the specific values the datatype covers, which are contained in the Substantive Value Domain for any variable. In the case of the Marital Status variable in the example above, the elements listed under the Substantive Value Domain make up the Value Space.
2. ***Characterizing Operations*** – the operations that distinguish one datatype from another. For example, an area allows for a perimeter and an area calculation, but a distance is just a linear measure.
3. ***Axioms*** – the rules one may assume the data obey. Confusingly, these are called *properties* in ISO/IEC 11404. One 11404 property all datatypes have is equality, and this, as we've seen above, is the defining characteristic that distinguishes data from terms or designations in general. Another example of an axiom is whether data are bounded or not. A finite list of numbers must be bounded, but the list of positive integers has no largest value. It is unbounded at the top.

The combination of axioms and characterizing operations for a value space determine a computational model associated with the data defined in the value space. By alluding to the need for defining equality for a concept defining a datum, we implicitly call into play an underlying computational model. Data are used for computation; and specifying the limits of that computation is what a datatype is for.

Second, ISO/IEC 11404 does not, and cannot, say what the value space is for the descriptions of datatypes contained there. So, the descriptions are generalized, and they address the axioms and characterizing operations that each kind of datatype, or datatype family, must have.

For instance, datatypes describing the marital status values above and genders (the set of gender categories and codes: {<m, male>, <f, female>}) have the same axioms and characterizing operations. The only difference is the value space. As such, these datatypes are both members of the same datatype family, namely the statistical type called nominal.

Nominal, Ordinal, Interval, Ratio, Quantitative, Qualitative, Discrete, and Continuous are names of datatype families typically used in the statistics. In the examples above, we make use of these.

4. Instance Variable

Moving further down the chain to data, we get to the Instance Variable. An Instance Variable is intended to be a variable used in a data set. For each data set, new Instance Variables are created.

The main addition in specificity is turning the sentinel categories into designations. Further, the list of sentinel values (designations) are managed in one set, the sentinel value domain. Separating the

substantive and sentinel value domains eases the burden on metadata management. Changes needed in one kind of value domain do not affect the other.

An example of the designations in a sentinel value domain is:

- <m, missing>
- <r, refused>

Since, the Instance Variable is associated with data in a data set, then the datatype of the data for that variable is necessary information as well.

Instance Variable (Inherits from Represented Variable)

Name	Marital Status	Age1
Population	US Deer Hunters in 2019	US Gun Shops in 2019
Sentinel Value Domain	<1, Missing> <2, Refused>	<1, Missing> <2, Refused>
Function	demographic	establishment
Physical Datatype	1-character	2-integer

The codes used to designate the sentinel categories are often determined by each statistical package. This topic will not be addressed in detail here.

The Physical Datatype addresses the kind of data as written on a file. The value $2.60 (two dollars and sixty cents) is often written as a real number with 2 decimal places. But monetary amounts don't follow all the rules for real numbers. The amounts at the third decimal place or after are truncated. The values are not rounded, as real numbers will be. This has an effect on computations, as the following example illustrates:

Take the average of $1.50, $1.30, and $1.00. The arithmetic average is $1.2666. The rounded real number average is $1.27, and the monetary, or scaled number, rounded average is $1.26. The reason is the fractional penny is dropped in the scaled situation. And the rules for scaled numbers correspond to how banks handle money.

Therefore, the physical datatype is often just an approximation of what is needed to describe values. Instead, it corresponds to how the values are written in a file. The actual use of the values depends more on the Intended Datatype at the Represented Variable level

F. Detailed Documentation for Foundational Metadata in DDI - CDI

Detailed documentation at for the Foundational Metadata in DDI – CDI can be found in this package in the folder: `\DDI-CDI Public Review 1\2 Model\Field-Level Doc\`.

III. Data Description

A. Introduction

The DDI-CDI model is defined as a Unified Modeling Language (UML) model. Figure 8 below shows a core portion of that model. The elements (classes) of the model appear as boxes with a name at the top and a list of properties below the name. Properties, listed in the bottom half of the box for the class, contain the payload of the class. Sometimes the value of a property is complex. The "definition" property of a Concept, for example, has the datatype of "InternationalStructuredString", which will have a text string, but also other properties such as whether it is translated and from what language it is translated. This complexity is the result of 20 years of incorporation of use cases into the model.

Classes may also have associations with other identifiable classes. In the diagram below a Datum has a simple association named "denotes" with a ConceptualValue. This relationship is read as "a Datum denotes a ConceptualValue". It is displayed in the diagram as an arrow that indicates the order in which the association is to be read. Classes that can be the target (object) of an association have a unique identifier and are reusable. The target end is indicated by an open arrowhead.

Some classes inherit from others. A ValueString inherits (is a type of) from an InstanceValue. This is indicated by the filled-in triangular arrowhead on the parent end.

Some associations indicate containership. A ConceptSystem aggregates (has) a set of Concepts. This is indicated by the diamond on the containing end of the relationship line and is read as "a ConceptSystem has Concept".

A Datum populates a cell of a dataset, database table etc.[3] through an InstanceValue. The Datum links some conceptual value to a physical representation. In the diagram below a ValueString is a physical representation. It future versions of the model, the physical representation could be an image, a sound byte or some other digital representation. Introduction of the ConceptualValue allows for the description of multiple representations in perhaps multiple platforms of the same measurement. A height, for example could be recorded as a decimal string or a binary string.

Physical structures (like files) are made up of DataPoints, each of which contains one InstanceValue.

In this model, InstanceVariables constrain DataPoints and Datums. They are no longer restricted to describing a column in a table.

The general idea in DDI-CDI is to be able to attach metadata at the "cellular" level, rather than at the structural level, and to allow those "cells" to be arranged into different structures without loss of descriptive information.

(For more information about how UML diagrams are used in DDI – CDI, please see the document "DDI Cross Domain Integration: Architecture and Alignment with Other Standards" in this package.)

This is how Datum is defined in the DDI4 prototype documentation and GSIM.docx

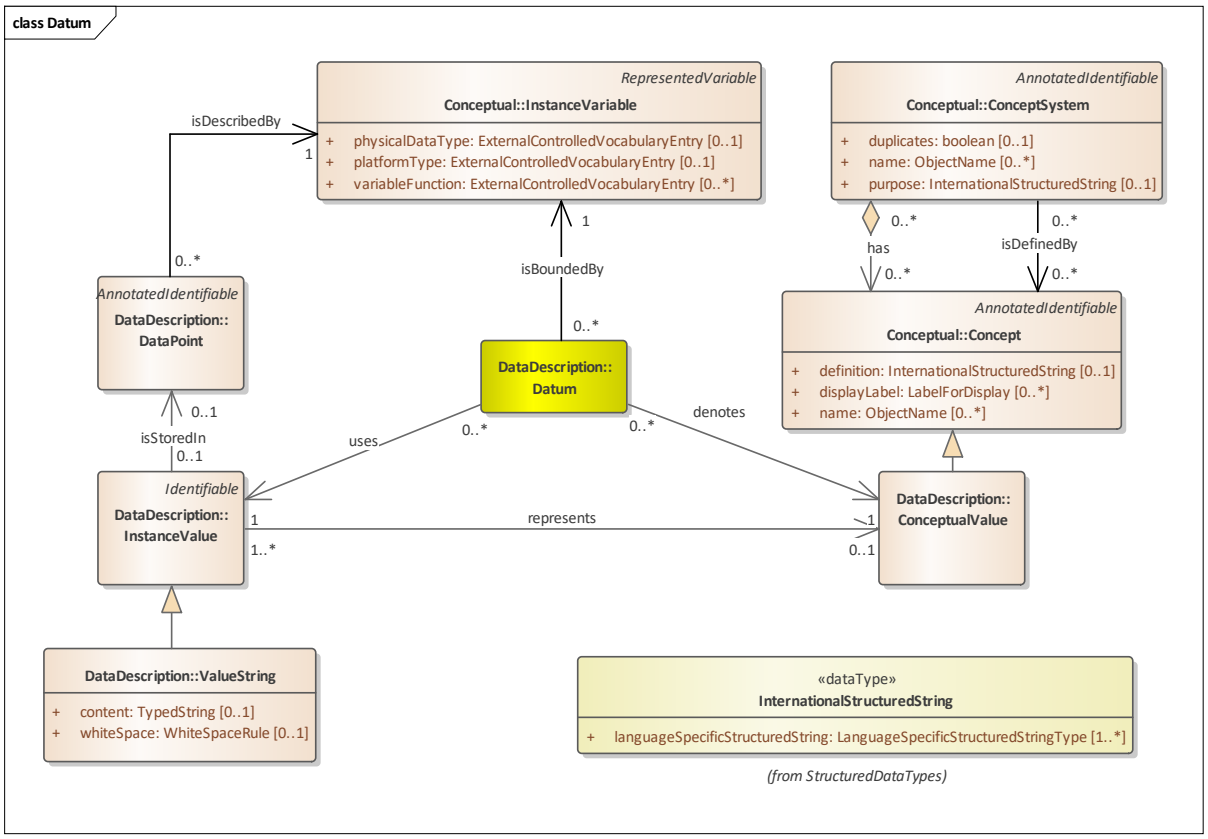


Figure 8: Datum in the DDI-CDI Model

B. Detailed Documentation for Foundational Metadata in DDI - CDI

Detailed documentation at for Data Description model in DDI – CDI can be found in this package in the folder: `\DDI-CDI Public Review 1\2 Model\Field-Level Doc\`.

C. Scope

The DDI-CDI Data Description provides the basis for describing a broad range of data structures using a core set of metadata elements. The model separates data structure and content in such a way as to allow data to be structured flexibly.

This section describes the general approach which DDI-CDI is taking, before going through the details for a selected set of data structures. The goal for DDI-CDI is to describe various data structures, both legacy structures such as rectangular data sets, multi-dimensional data, and event data, but also new ones like data streams or data lakes. The approach is independent of any specific domain or discipline, as similar data structures are used broadly in a range of research settings.

The model has structures for documenting different data structures and the transformations between them.

Data structures are a way to organize data in an organized way in order to be processed by software programs. The current DDI-CDI model can be used to describe data from different data structures using a Datum-based approach. This approach involves describing each “cell” in a granular fashion, such that the same values can be recognized when occurring in differently structured data sets.

The following structures are covered:

Wide Data: Traditional rectangular unit record data sets. Each record has a unit identifier and a set of measures for the same unit.

Long Data: Each record has a unit identifier and a set of measures but there may be multiple records for any given unit. The structure is used for many different data types, for example event data and spell data.

Multi-Dimensional Data: Data in which observations are identified using a set of dimensions. Examples are multi-dimensional cubes and time series. (Note that support is provided for time-series-specific constructs to support some legacy systems which are not based around the manipulation of multi-dimensional data “cubes”.)

Key-Value Data: A set of measures, each paired with an identifier, suited to describing No SQL and Big Data systems.

Each of the four data structure types - Wide, Long, Dimensional, and Key-Value - are structured in slightly different ways but share some common features. Before going into how each of them can be structured a set of related core components will be presented, applicable across the range of data structures described.

This chapter describes the DDI-CDI approach to each of the above-mentioned data structures according to the following outline:

A. Introduction

B. Detailed Documentation for Foundational Metadata in DDI – CDI

C. Scope

D. Basic Concepts
   1. Variables and Values
   2. Keys
   3. Data Structure Components

E. Wide Format (Unit Record Data Structure)
   1. Example
   2. Discussion of Structure and Diagrams – Wide

F. Long Data Format
   1. Example
   2. Discussion of structure and diagrams – Long

G. Multi-Dimensional Format
   1. Example
   2. Discussion of structure and diagrams – Dimensional

H. Key-Value Format
   1. Example
   2. Discussion of structure and diagrams – Key-Value

H. Physical Data Set (Wide Format)

I. Transformations between Formats/Examples
   1. Wide and Long: Correspondence between Unit record data and data in a Long format
   2. Wide and dimensional: Unit record data tabulated into an aggregate data Cube
   3. Long and Dimensional: Dimensional data represented in a Long data format
   4. Key-Value and Wide: Key-Value Stores in RAIRD
   5. Time Series
   6. Key-Value Stores and Streams

D. Basic Concepts

Before explaining about the four data structure types some basic shared concepts require explanation.

1. Variables and Values

Consider this portion of the model:

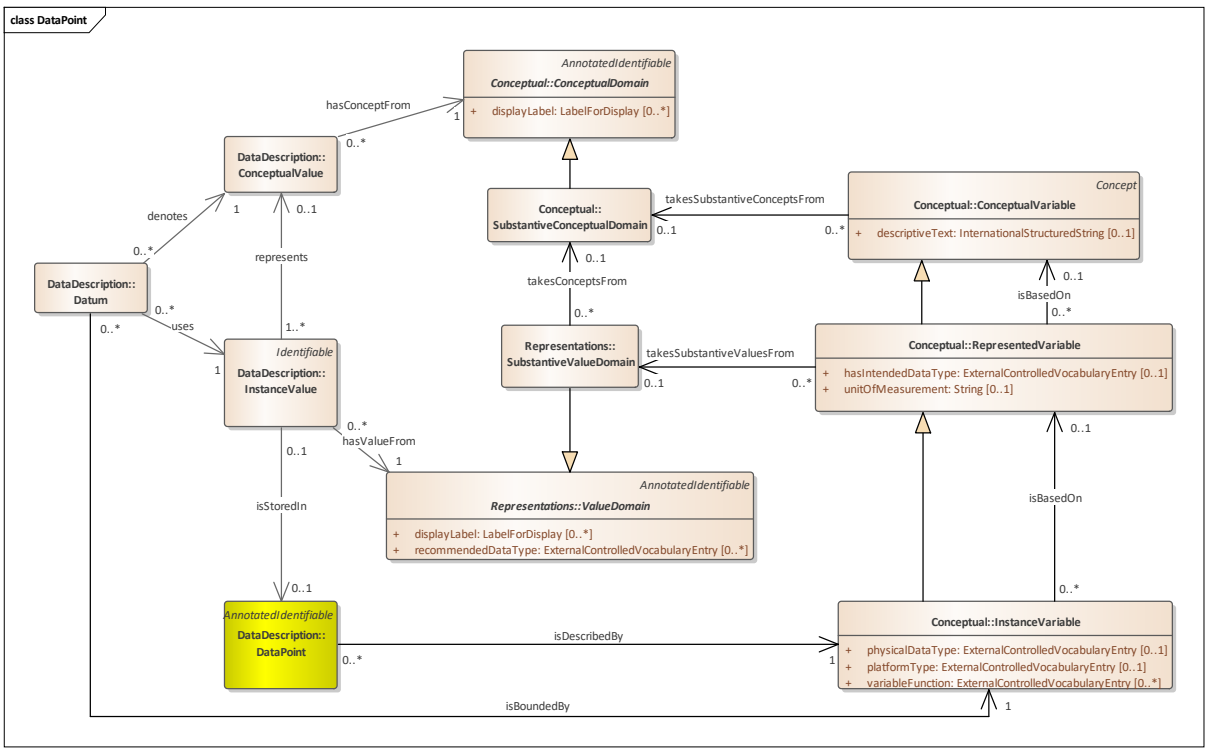


Figure 9: DataPoint in the model

To the right in Figure 9 above the variable structure that provides meaning to the data is displayed. (The variable structure is often referred to as the Variable Cascade, as described earlier in this document in the section on Foundational Metadata.)

The Variable Cascade facilitates distinctions between Instance Variables (the variable as in the dataset), Represented Variables (the reusable components of a variable) and Conceptual Variables that expresses the conceptual basis of a variable.

In the middle of the figure domains are displayed, which provide representations and contexts for the data values. The SubstantiveConceptualDomain specifies the set of valid concepts for the ConceptualVariable, while the SubstantiveValueDomain specifies the set of values for the corresponding InstanceValues. ConceptualDomain and ValueDomain are abstract classes of which SubstantiveConceptualDomain and SubstantiveValueDomain are sub-classes.

The left part of the figure displays how data values are modeled. The DataPoint represents a 'cell' in a data structure that stores the InstanceValue, which is the actual representation of the observation or value in a data set. Datum is a mediator between the InstanceValue and the ConceptualValue, which is a more generic and reusable value format.

The DDI 4 Data Description is based at the core on the description of a single datum associated with a data cell (a DataPoint). One example of a DataPoint is a single cell in a rectangular table. DataPoints, in turn, are organized into structures. (This will be described further below.)

Earlier versions of DDI (e.g., the DDI Codebook and DDI Lifecycle specifications) allowed for the description of two structures, records and NCubes. The record structure was defined in terms of a list of variables and did not allow for the description of individual cells. The NCube structure was an assembly of variables, to allow for the process of tabulation to be recognized. DDI – CDI takes a more granular and more generalized approach.

DDI-CDI also explicitly models the conceptual value that is represented by the symbol in the DataPoint along with the Unit measured by the value associated with a DataPoint. The DDI-CDI approach allows for the explicit description of different representations of the same measured value. This is called Instance Value in the model. A measurement (the captured value) of Marie's (the Unit) longevity (the type of measure) at a specific time might, for example, be represented in Arabic numerals, roman numerals, or words. In our unit record example Marie's longevity is recorded in Arabic numerals as "73.7". The measurement might be contained in different software packages that have subtly different representations of a number. All of these represent the conceptual (measured) value of the longevity but use different symbols to do so.

Following GSIM, DDI-CDI includes the elements *Datum*, and *DataPoint*. In the DDI-CDI model, a Datum connects a conceptual value (a type of Concept) with a representation (a perceivable symbol, the instance value). In the case of Joe's height taken on 2019-07-16, a Datum would link the conceptual value of that height to a specific sequence of characters (e.g. "183,5" cm). Another Datum might link

that same conceptual value to a different sequence (“183.5” cm). Both instance values stand for the same measurement but use different physical symbols.

A DataPoint is a kind of container for the representation of a value, the instance value. Think of a cell in a spreadsheet. Its column corresponds to an InstanceVariable. Its row corresponds to a Unit. It may have content or it may be empty. Changing the format of the cell changes the representation (instance value), but the underlying value (conceptual value) remains the same.

This allows the single Datum to be be ‘followed’ across different data structures. This differs from approaches used in many other similat models, including other DDI products (e.g., DDI Codebook, DDI Lifecycle) where some of this information was attached at a higher level (typically the data set or record). DDI-CDI is a little more explicit than GSIM in describing the conceptual value and its representation, the instance value. As an information model, GSIM does not deal with the details of physical representation of data. DDI-CDI needs to be able to describe representation in more detail.

A Datum populates a cell of a dataset, database table etc. The general idea in DDI-CDI is to be able to attach all necessary metadata to the single Datum so that it can be ‘followed’ across different data structures. This differs from some other approaches used in other DDI products (DDI Codebook, DDI Lifecycle) where some of this information was attached at a higher level (eg, the data set or record).

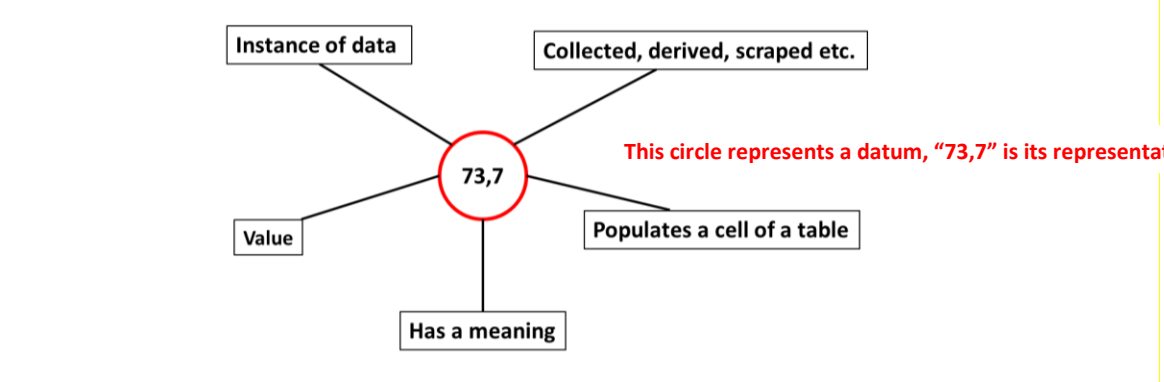


Figure 10: Datum and its connotations

2. Keys

Another central concept for Data Description is that of the Key. In the model a Key is used for the identification of data and may comprise a set of Key members or be a unitary value. Figure 11 below shows how an InstanceValue is linked to a Unit (an individual or object of interest) via a Key that identifies the DataPoint where the InstanceValue is stored.

A Key is defined as a collection of data instances that uniquely identify one or more data points. A KeyMember is a single data instance that is a part of an aggregate key.

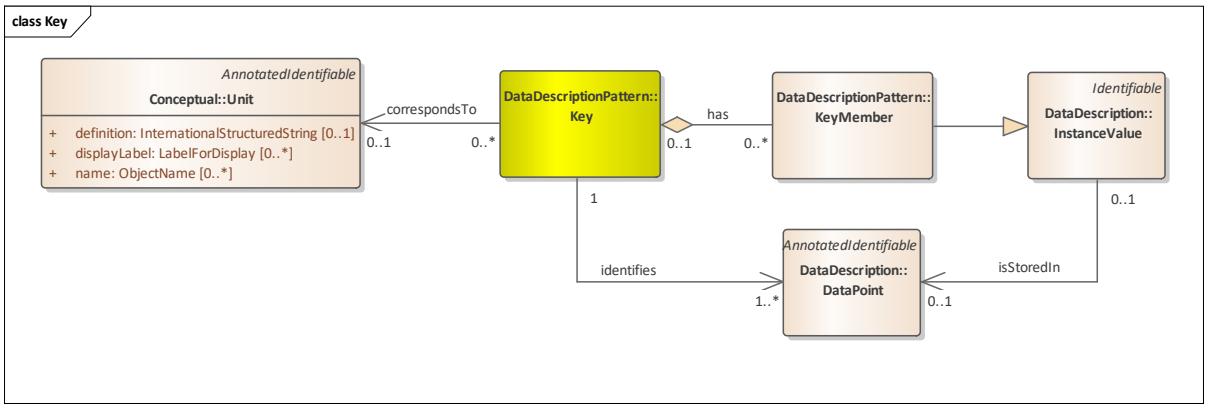


Figure 11: Key

3. Data Structure Components

A third feature are DataStructureComponents, allowing the RepresentedVariables to take different roles. Each data structure type - Wide, Long, Multi-Dimensional and Key-Value - has its own set of DataStructureComponents, with some being common across data structure types. The components which reflect roles are the IdentifierComponent, AttributeComponent and MeasureComponent. The roles allow a RepresentedVariable to serve as a Measure in one context and as an Identifier in a different context, for example. This will be detailed below in the description of the different data structure types.

Roles allow users to assign different functions to variables according to their context of use. Roles are not inherent in variables but can be imposed on them. In DDI-CDI there are currently three roles:

- **Identifier** - An identifier role that serves to differentiate one record from another. More than one variable may be used in combination to produce a compound identifier.
- **Measure** – Variables tagged with the measure role represent the values of interest.
- **Attribute** – The attribute role serves to provide information about the measures of interest. Variables might, for example, describe the conditions of a measurement. This way attributes can be used to link metadata or paradata to the Measure of interest.

A variable may take on different roles in different contexts.

E. Wide Format (Unit Record Data Structure)

1. Example

A Unit Record data table, as shown in Figure 12, is a common way to organize data. Each record has a set of observations about a single unit. The record has a unit identifier and a set of measures and/or descriptors which are the same for each unit. The unit identifier can be used as an identifier for the record, because each unit has only one. This structure is also referred to as a rectangular data file.

PersonID	Sex	Born	Died	RefArea	Longevity
Marie	Female	3.3.1932	12.1.2005	Newport	73.7
Henry	Male	8.1.1929	6.2.2008	Cardiff	78.8
etc.					

Figure 12: Unit record data table

The objects of the Wide format Unit Record data table are UnitDataRecords, Variables and InstanceValues. In the Wide format the rows correspond to each unit record, which is a set of InstanceValues for one entity. The columns correspond to each variable measure or categorization. Cell entries are InstanceValues.

A cell in the Unit record table is an intersection between a column representing a variable and a row representing a measurement unit. See for example '8.1.1929' in Figure 13 (yellow highlighting).

Each cell of the table contains an InstanceValue. 'Marie' and 'Henry' (green highlighting) are identifiers for each of the records. 'Sex', 'Longevity' etc. are variables (blue) and 'Female' and '78.8' are example of InstanceValues (red).

PersonID	Sex	Born	Died	RefArea	Longevity
Marie	Female	3.3.1932	12.1.2005	Newport	73.7
Henry	Male	8.1.1929	6.2.2008	Cardiff	78.8
etc.					

Figure 13: Wide format object

The WideDataSet contains DataPoints, all the 'cells' in the table. Columns are variables, and each row contain the DataPoints for one Unit. Some of the DataPoints contain values keys that identify the DataPoints common to an individual row of the table. A WideKey can have more than one Member - e.g. more DataPoints which act as identifiers. This will be further explained below.

2. Discussion of Structure and Diagrams – Wide

A Wide table row is further structured by three DataStructureComponents types:

- IdentifierComponents - the DataPoints which serve to identify the row.
- MeasurementComponents - the DataPoints in each row which contain the measures of interest.

- AttributeComponents - DataPoints which provide context for the MeasureComponents.

RepresentedVariables provide SubstantiveValues for a WideKeyMember.

In the example dataset displayed in Figure 14 below the "PersonID" column contains DataPoints that contain the key values that identify a row and also correspond to a Unit.

The DataPoint in the upper left of the table contains the key value "Marie". That DataPoint identifies the other DataPoints also associated with the person named "Marie", the DataPoints in the first row of the table.

A WideKey can be composed of more than one WideKeyMembers. Our table might have, for instance, have contained another column like "Family" so that we could identify the Marie in a particular family. (This might be important in a data set which had more than one unit named "Marie".)

A row of the table is also further structured by DataStructureComponents.

These are defined by RepresentedVariables, which in turn provide the SubstantiveValueDomain (often a Codelist) for a WideKeyMember.

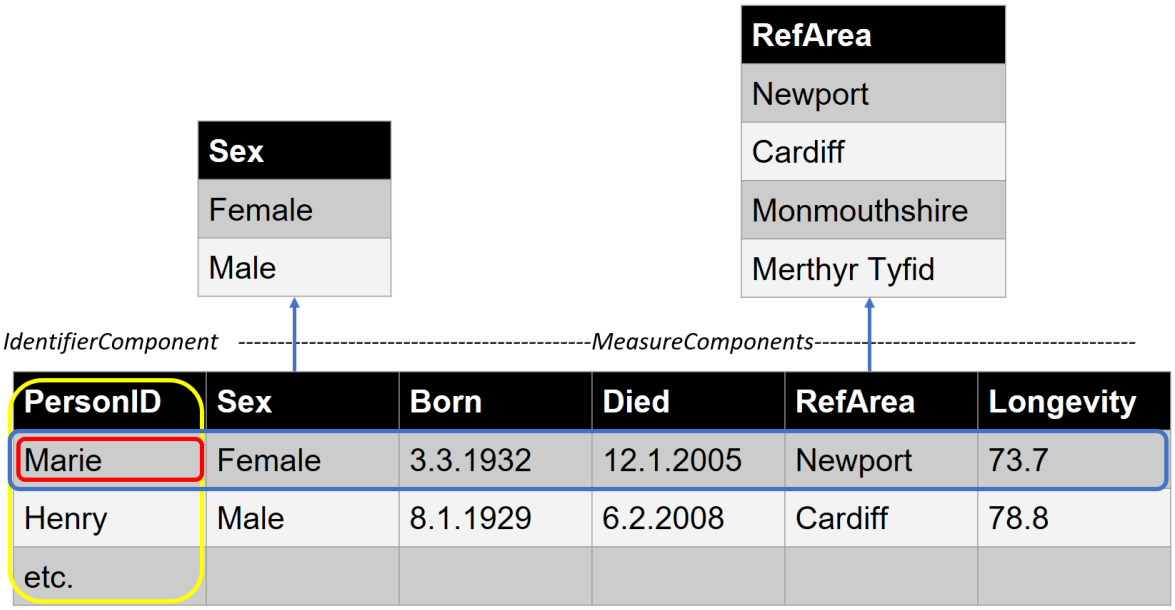


Figure 14: Wide structure

In the figure above, PersonID is an identifier for a person, Sex, Born, Died, and Longevity and RefArea are the measures of interest.

These roles are not fixed. For another purpose, RefArea might be considered an attribute of the measures. Roles are often slightly different when the same data is viewed using different formats (PersonID is the only identifier needed for the Unit Record format above – when expressed in a Long

format, it would be only one needed component of a compound identifier – more than one variable would take on the role of identifier. (See RAIRD example, below).

The diagram in Figure 15 below shows the DDI-CDI classes used to represent unit data in wide format. This is probably the most common layout for data – the traditional table of data as used in many statistical packages and spreadsheet programs. Columns are variables and each row contain the DataPoints for one Unit.

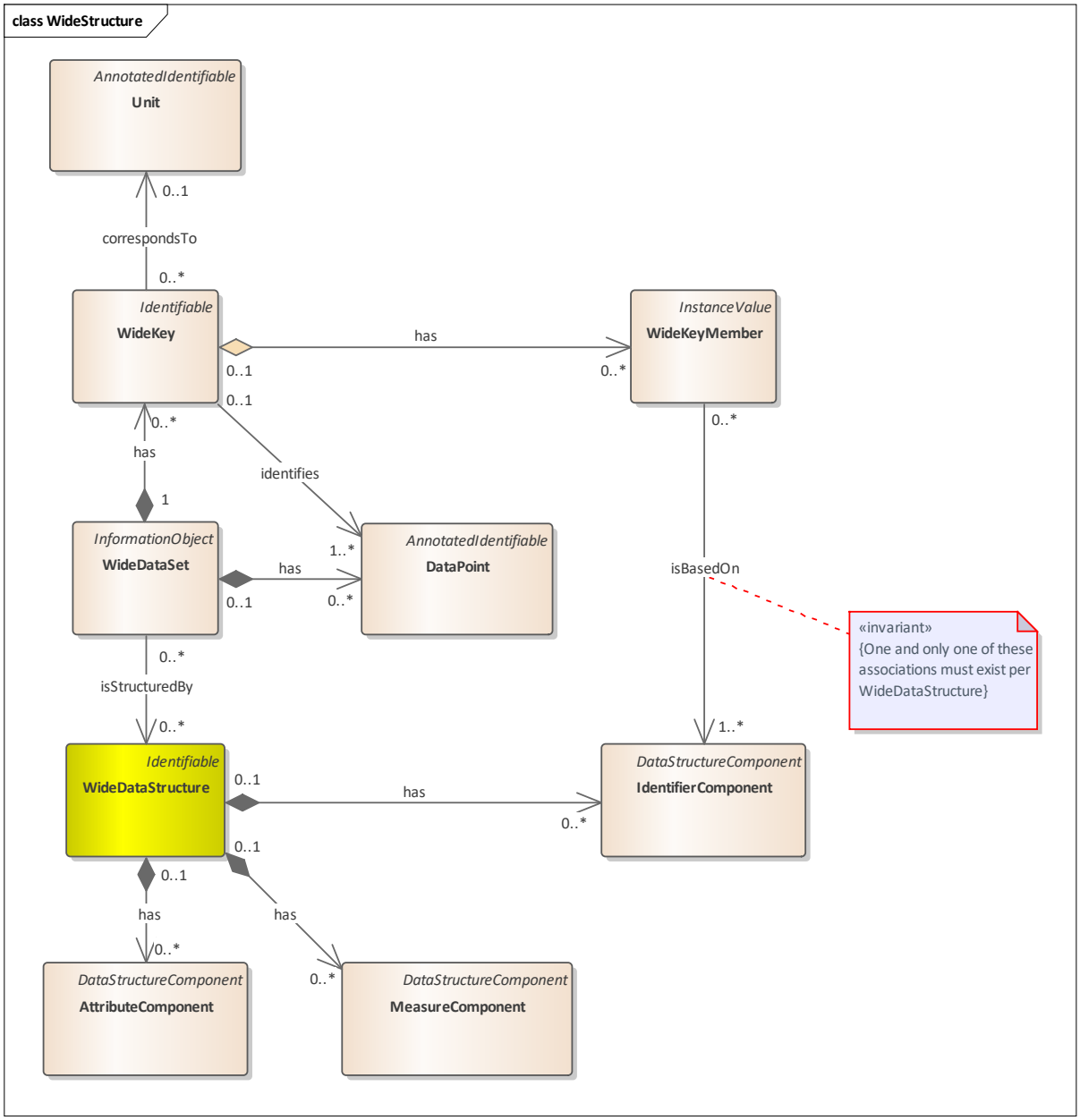


Figure 15: WideStructure

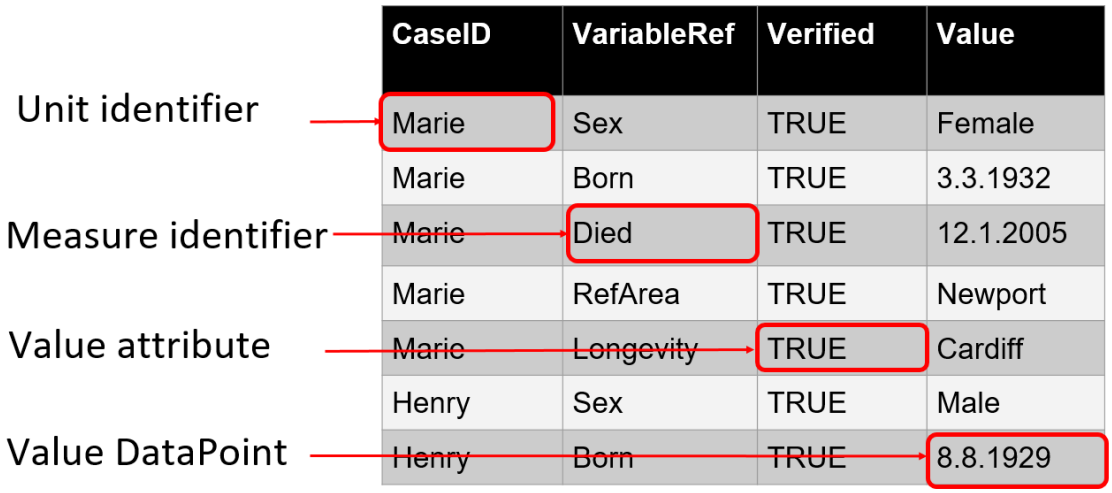
F. Long Data Format

1. Example

The same data as in the Wide example can be expressed in a different format called Long as shown in Figure 16 below. This format is often used to express event data.

In the Long format columns correspond to each kind of object in a Wide (unit record) description. Each row now contains a unit identifier, a variable identifier, and a data point with an instance value.

The rows correspond to each value of each (non-identifying) variable for each Wide record.



CaseID	VariableRef	Verified	Value
Marie	Sex	TRUE	Female
Marie	Born	TRUE	3.3.1932
Marie	Died	TRUE	12.1.2005
Marie	RefArea	TRUE	Newport
Marie	Longevity	TRUE	Cardiff
Henry	Sex	TRUE	Male
Henry	Born	TRUE	8.8.1929

Figure 16: Long format

In pure form, each row of a long structure contains a DataPoint with the value of interest (the instance value) along with identifiers for a unit and a column with a code that identifies the variable (VariableRef above) that associates with the value in the value DataPoint. In the figure above the Value column contains DataPoints with values from more than one variable, Sex, Born, Died, RefArea, and Longevity. Note that there may be many rows for a unit (like for "Marie"). There can also be columns containing attribute values. The "Verified" column is an attribute that indicates whether the value in the Value column has been verified.

CaseID	VariableRef	Value
Marie	Sex	Female
Marie	Born	3.3.1932
Marie	Died	12.1.2005
Marie	RefArea	Newport
Marie	Longevity	73.7
Henry	Born	8.1.1929
Henry	Died	6.2.2008
Etc.		

Here we can see how a complete record for one unit from our Wide example might be represented: each column in the Wide format for a single row becomes a row in the Long format (see Transformations between Data Structures, Examples, below).

2. Discussion of structure and diagrams – Long

The high-level view of the long model is shown below. Each DataPoint in the dataset is based on one of five data structure components. Each component is associated with a RepresentedVariable that can define a column in the tall table.

These perform the following functions:

• IdentifierComponent – one of possibly several components that together identify the Unit associated with the measures and attributes. In the example above this is the CaseID column in Figure 16.

• MeasureComponent – a measure just like in the wide layout. This allows a hybrid wide-tall layout. There is no such column in the example above if there were the values for the Marie rows in Figure 16 would all be the same.

• AttributeComponent – an attribute that annotates the associated measure values. This is the Verified column in Figure 16.

• VariableDescriptorComponent – an indicator of the InstanceVariable in each associated VariableValueComponent DataPoint (see Diagram). This is the VariableRef column above. In the first row the code “Sex” indicates that the value “Female” is associated with the variable named “Sex” used in the Wide table. Note that this component has an association to a specific VariableValueComponent.

• VariableValueComponent – defines a column that has a value associated with the value in the VariableDescriptorComponent. This is the Value column above. The “3.3.1932” is interpreted as the date that Marie was born. This column will have to have a datatype as generic as needed to hold all of the values from the set of variables indicated in the VariableDescriptorComponent. In the example above there is a mix of numeric (Longevity), Date (Born, Died), character (Sex), and geographic codes (RefArea) variables. A character datatype for the associated RepresentedVariable would be required. In many statistical platforms there are tools to reshape data between wide and long format. Many have restrictions that would force all of the measure values to have the same datatype (e.g. all numeric).

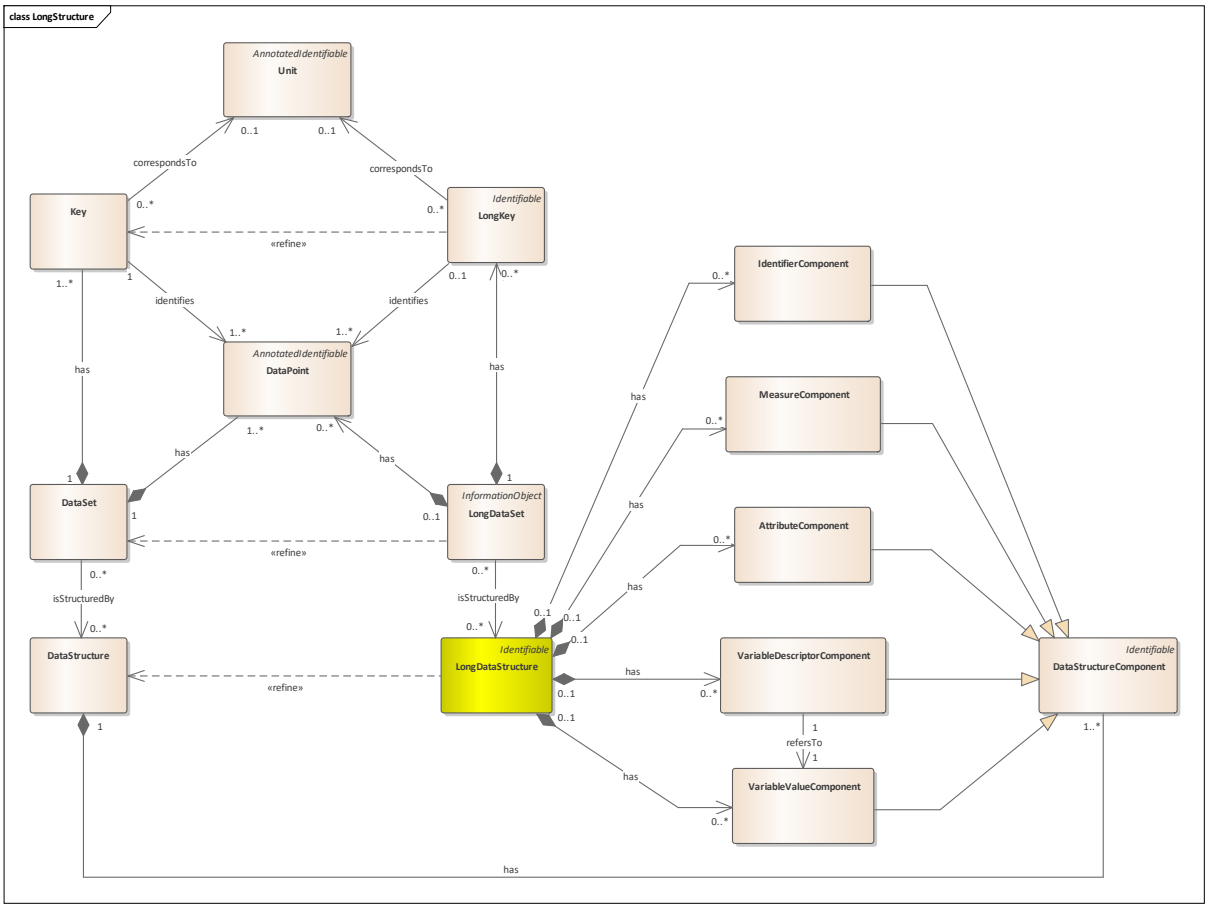


Figure 17: LongStructure – overall diagram

On the right side of Figure 17 we the different data structure components described above are shown.

The left side displays how the Long Key identifies the DataPoint and brings it together with the other components.

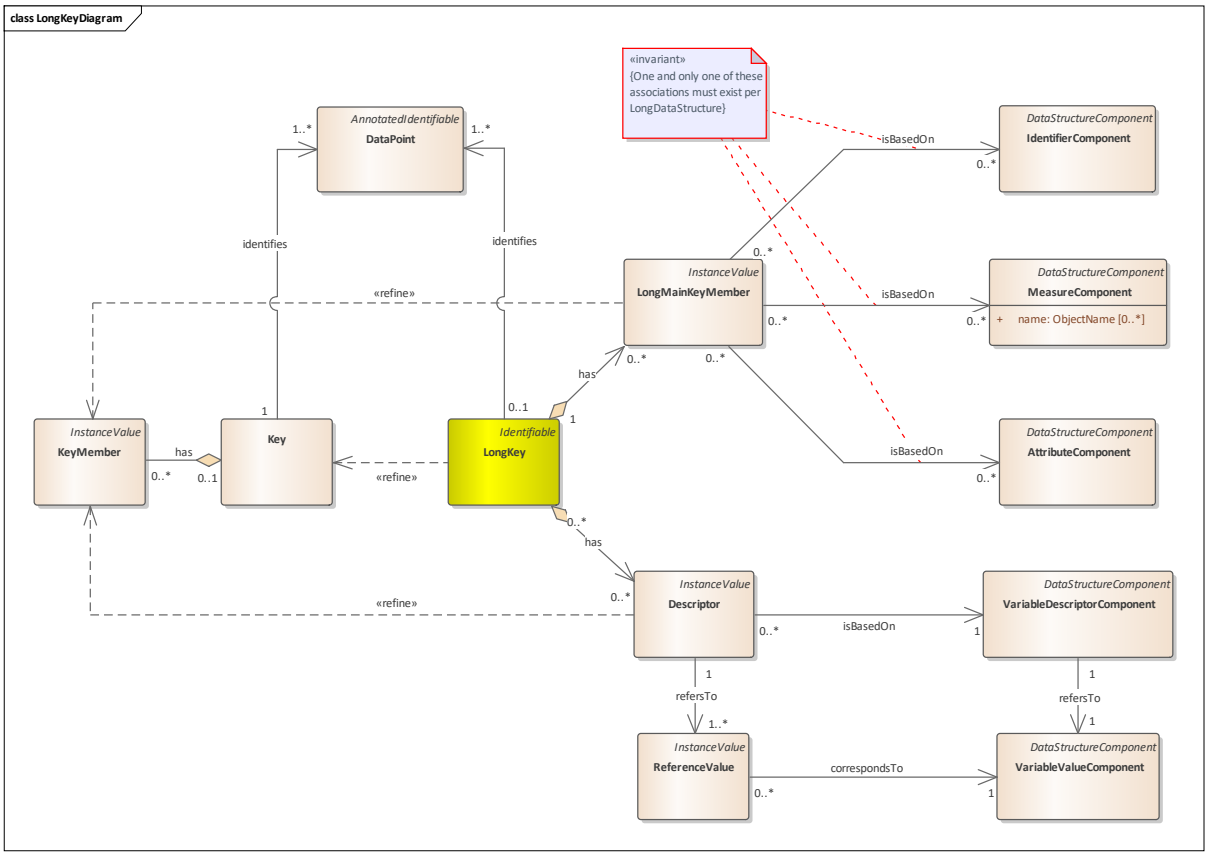
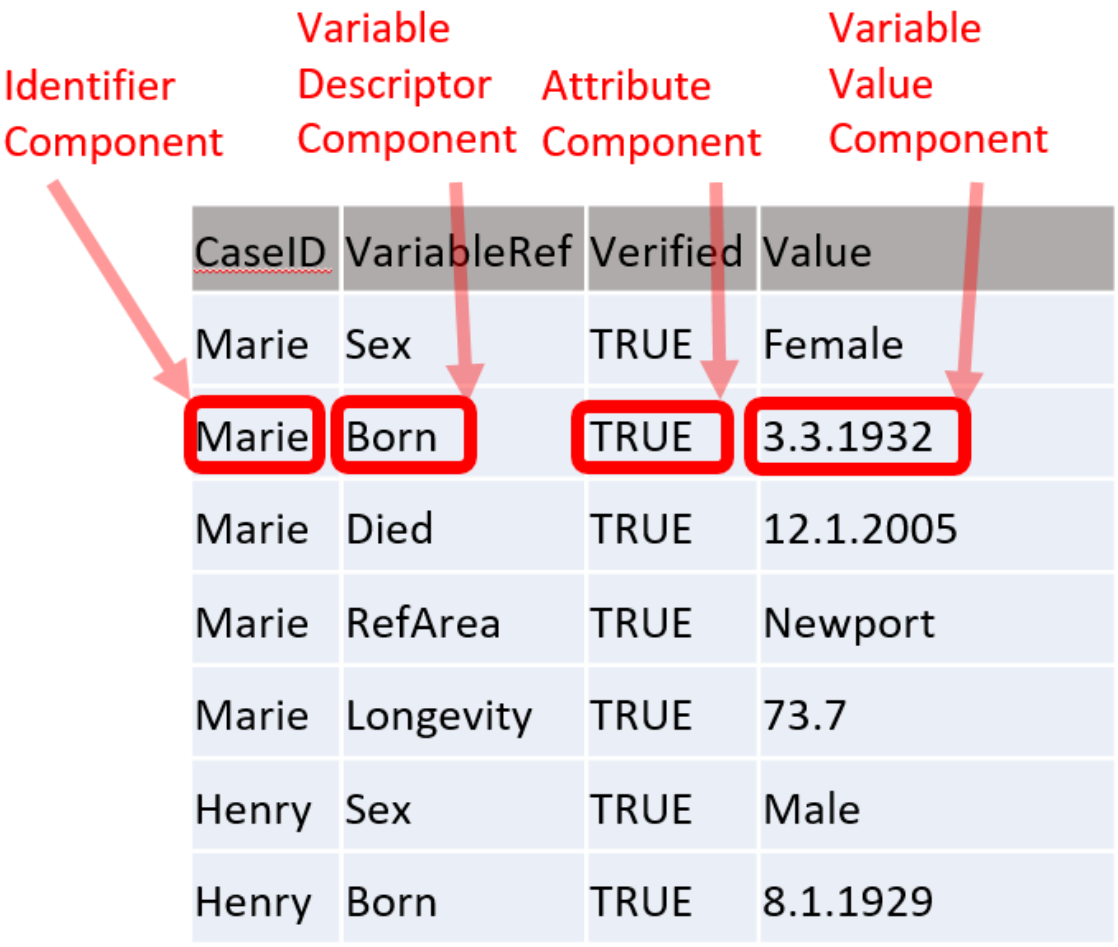


Figure 18: Long format – LongKey diagram

The diagram in Figure 17 conceals some of the complexity involving the association between the LongKey and the LongKeyMember. The LongKey (Figure 18) is actually a composite of LongKeyMembers, each of which is based on one of the five component types. A LongKey could include, for example, two IdentifierComponents, such as Household and personInHousehold.

The long layout brings out the utility of the Datum based approach and the use of keys to describe data. In the long dataset example seen below the values of the “Value” column are in a different conceptual domain in each row. A traditional variable having one conceptual and one value domain makes no sense for the column.

The VariableRef column contains the VariableDescriptorComponents of the compound key describing the InstanceValue in each row of the value column. The column VariableRef itself is a DescriptorVariable that can be described as having codes that point to InstanceVariables. In the highlighted cell in that column “Born” is a code for an InstanceVariable that describes dates of birth. The other two columns are associated with InstanceVariables that could appear in a wide layout. CaseID contains id values each of which is an IdentifierComponent of the compound key. Verified contains AttributeComponents of the key. Together the compound key of “Marie”, “Born”, and “TRUE” provides context for the highlighted InstanceValue of “3.3.1992”. They allow it to associate it with the “3.3.1992” in the “Marie” row of the “Born” column of the wide example table above.



CaseID	VariableRef	Verified	Value
Marie	Sex	TRUE	Female
Marie	Born	TRUE	3.3.1932
Marie	Died	TRUE	12.1.2005
Marie	RefArea	TRUE	Newport
Marie	Longevity	TRUE	73.7
Henry	Sex	TRUE	Male
Henry	Born	TRUE	8.1.1929

Figure 19: Long table components

The VariableDescriptorComponent diagram below shows how the VariableDescriptorComponent relates to other components of the model.

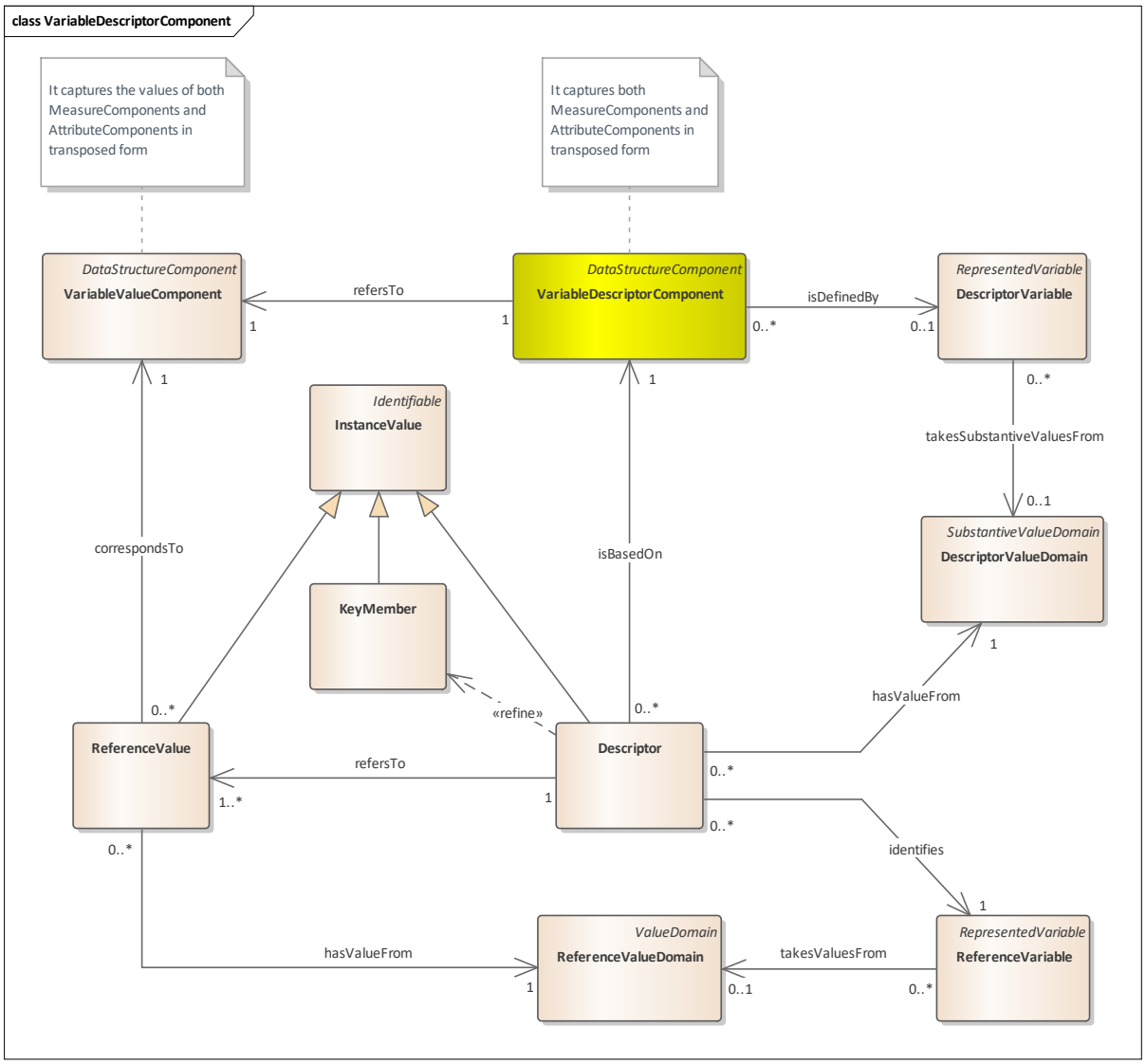


Figure 20: VariableDescriptorComponent diagram

G. Multi-Dimensional Format

1. Example

Sometimes data are presented in dimensional form. In the example below (Figure 21) there are three dimensions: geographic, with Categories of Newport, Cardiff, Monmouthshire, and Merthyr Tydfil.; temporal, with categories like 2004-2006; and gender, with categories of Male and Female. The numeric values in the cells are often aggregates computed on some variable or combination of variables, in this case the mean of longevity. Cells might also contain direct measurements such as with data from an experiment with a factorial design. Dimensional data are commonly displayed in a dimensional table like a pivot table.

	2004 - 2006		2005 - 2007		2006 - 2008	
	Male	Female	Male	Female	Male	Female
Newport	76.7	80.7	77.1	80.9	77.0	81.5
Cardiff	78.7	83.3	78.6	83.7	78.7	83.4
Monmouthshire	76.6	81.3	76.5	81.5	76.6	81.7
Merthyr Tyfid	75.5	79.1	75.5	79.4	74.9	70.6

Figure 21: Dimensional data presented in tabular form

2. Discussion of structure and diagrams – Dimensional

A cube is a multi-dimensional array of cells (DataPoints). Values in the cells may be the result of an aggregate computation or a direct measurement.

At a logical level the structure of the cube is defined by a set of Dimensions (the DimensionalDataStructure in the diagram below).

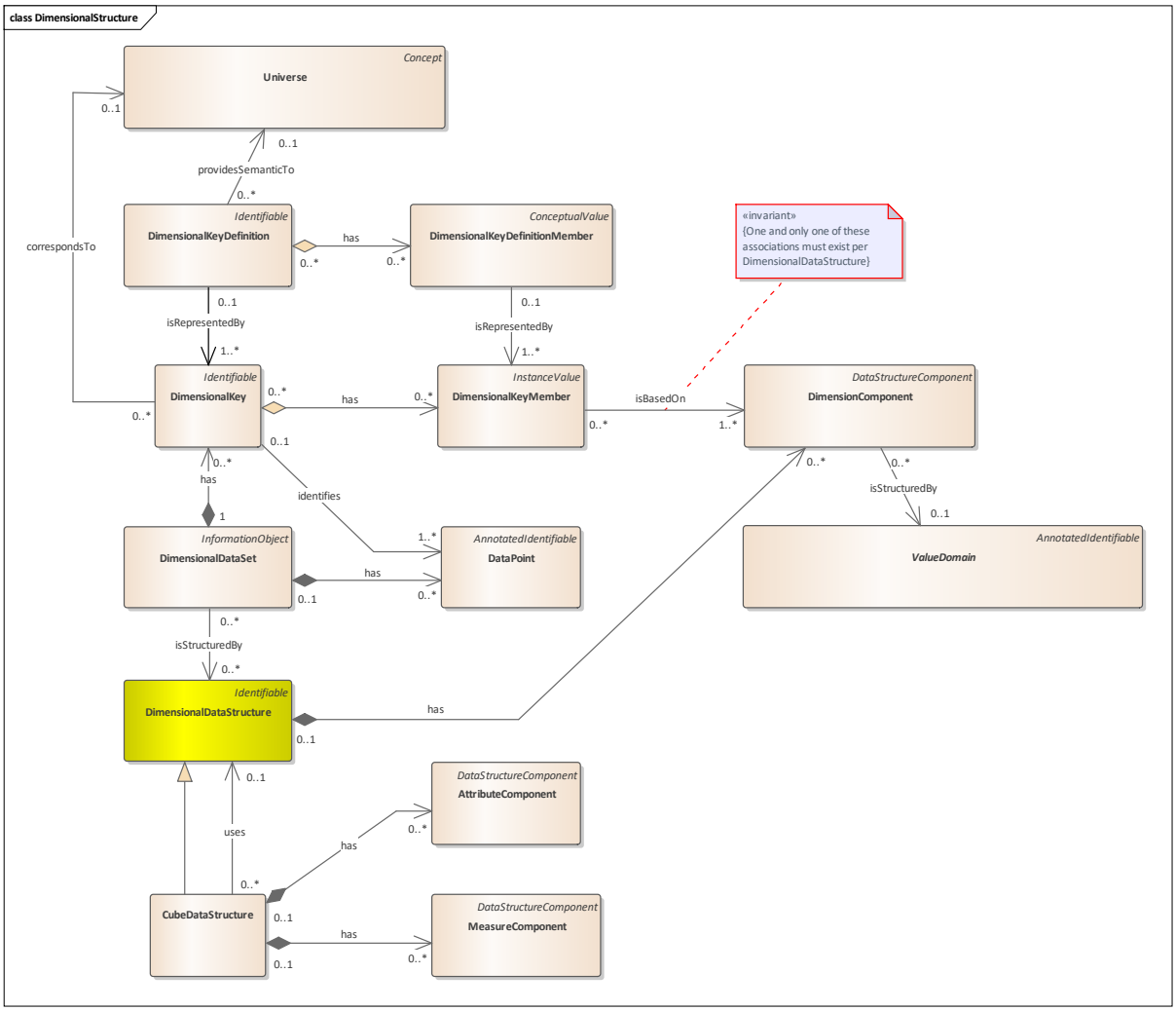


Figure 22: DimensionalStructure - details

Each dimension (DimensionComponent) is, in turn structured by a SubstantiveValueDomain and defined by a RepresentedVariable. The latter also brings along the specification of a Universe and a Concept. A Dimension can be categorical, for example ("Male", "Female"). In this case the SubstantiveValueDomain would consist of a Codelist. Typically cubes containing aggregate data would have primarily (or only) categorical dimensions. A DimensionComponent might also have a described value domain. Experimental data might, for instance, employ an independent variable measured as a real number (e.g. person's weight).

While there may be some underlying continuous variable for a Dimension, a Dimension may often be delineated by discrete dimensional categories. Time, for example, is a continuous measure. In our cube example, though, it has been transformed into a set of three-year categories like 2004-2006. This, along with the other two dimensions (gender, and geography), allows for the delineation of discrete cells in a table. Note that the time periods in this example (Figure 22) overlap.

The DimensionalComponents form the basis for keys. A DimensionalKey is a composite of one value from each SubstantiveValueDomain of a DimensionComponent. This composite DimensionalKey

identifies the location of a DataPoint in the dimensional structure. Our example cube, for example, contains mean longevity data measured on people of Wales. The DataPoint (cell) identified by the key value (2006 – 2008, Female, Newport) is associated with that subset of people.

Partial Keys – in which only a subset of the DimensionalKeyMembers have values specified for them – can be used to refer to regions (or "slices") within the cube. (DDI-CDI does not explicitly model this; it is left to implementations to handle partial Keys if this is useful or required.)

Each DimensionalKeyMember (InstanceValue) of the DimensionalKey is also associated with the concept 'Male' in a ConceptualValueDomain. This would provide meaning for the DimensionalKeyMember in the case of an aggregate data set.

Categories within the Dimension may be additive or not. In our example the geographic areas could be combined to create larger areas. The year range categories could not be combined in a straightforward fashion given that they overlap.

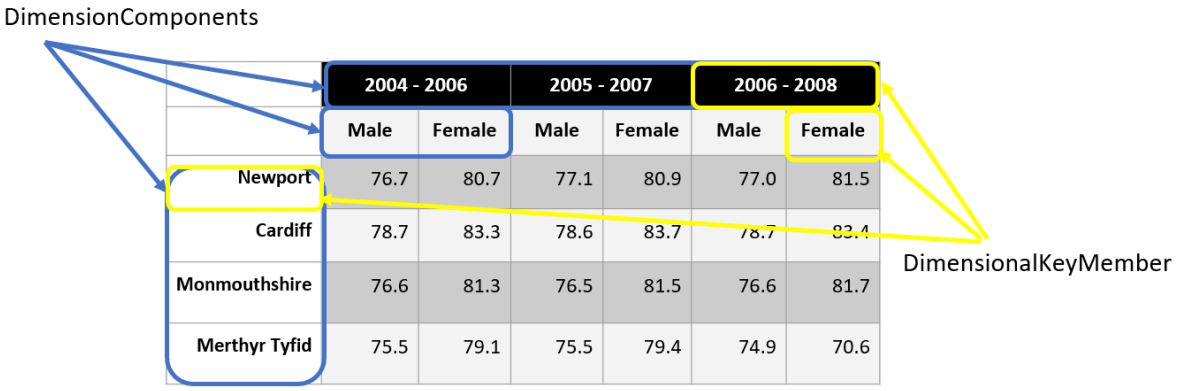


Figure 22: DimensionComponents and DimensionalKeyMembers

In addition to structure a cube has content. The CubeDataStructure also includes a MeasureComponent and an AttributeComponent. The MeasureComponent is defined by a variable for that value.

A QualifiedMeasure as the measure for the whole cube (e.g. mean of longevity), while a ScopedMeasure is for each cell in a cube as its Population narrows the Universe of the Qualified measure

There might also be Attributes associated with each cell in a cube. One example of an attribute might indicate whether the measure for the cell was imputed.

The DDI-CDI model bundles a number of information elements into an Instance Variable. While a cube like our example may have a measure with a single concept, each cell in the cube has a different Population. The upper left cell in the example has a mean of longevity for Males in Newport in 2004-2006. The cell just to the right of it has mean of longevity for Females in Newport in 2004-2006. The DDI-CDI Dimensional model includes the notion of a ScopedMeasure for the InstanceVarriable for each cell in a cube and a QualifiedMeasure as the measure for the whole cube. The ScopedMeasure has a Population which narrows the Universe for the QualifiedMeasure. This diagram (Figure 23) shows how those fit into the model.

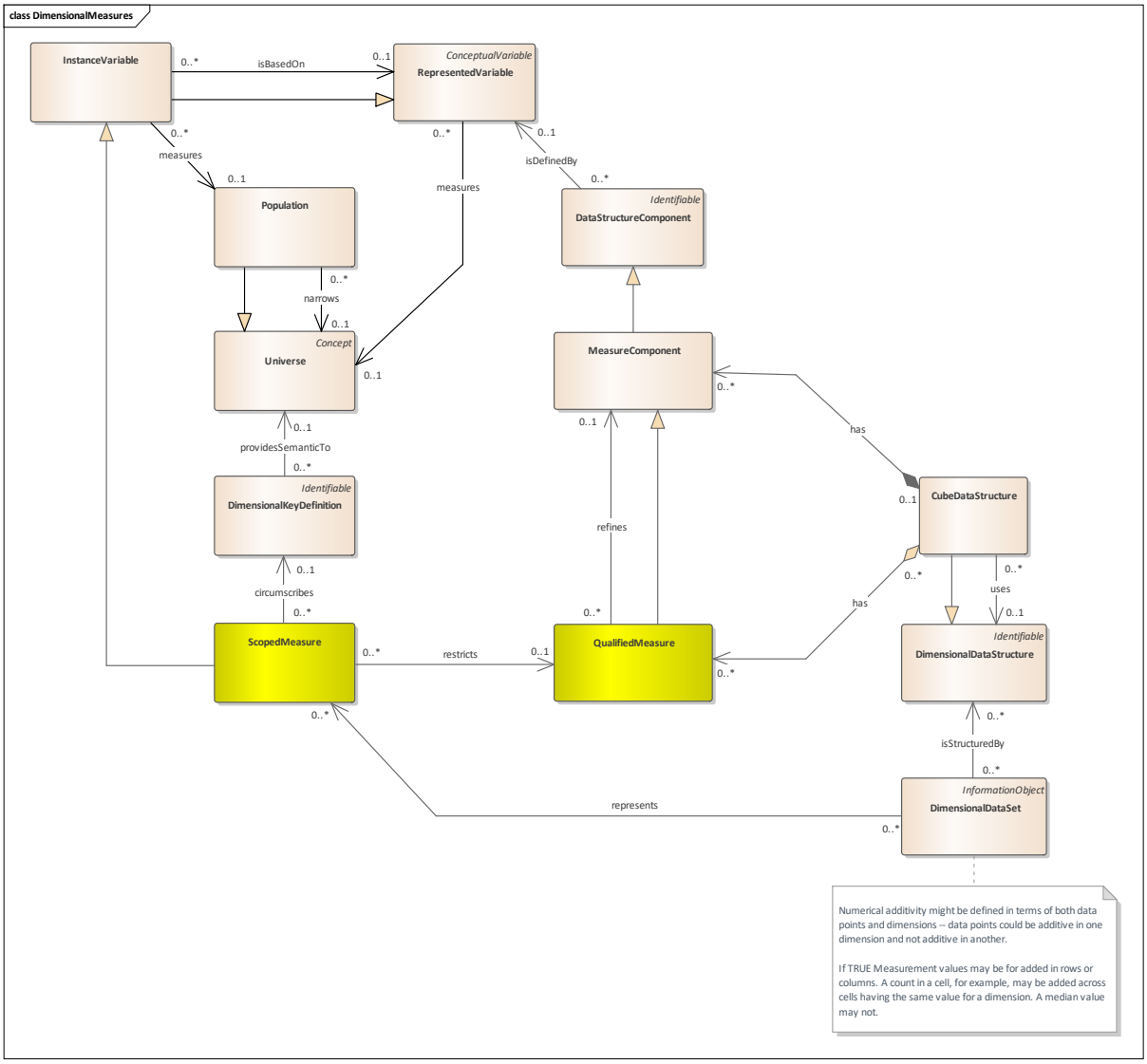


Figure 23: DimensionalMeasures

Each cell in the cube's DimensionalDataStructure can have not only measures associated with it, but also attributes (see diagram below).

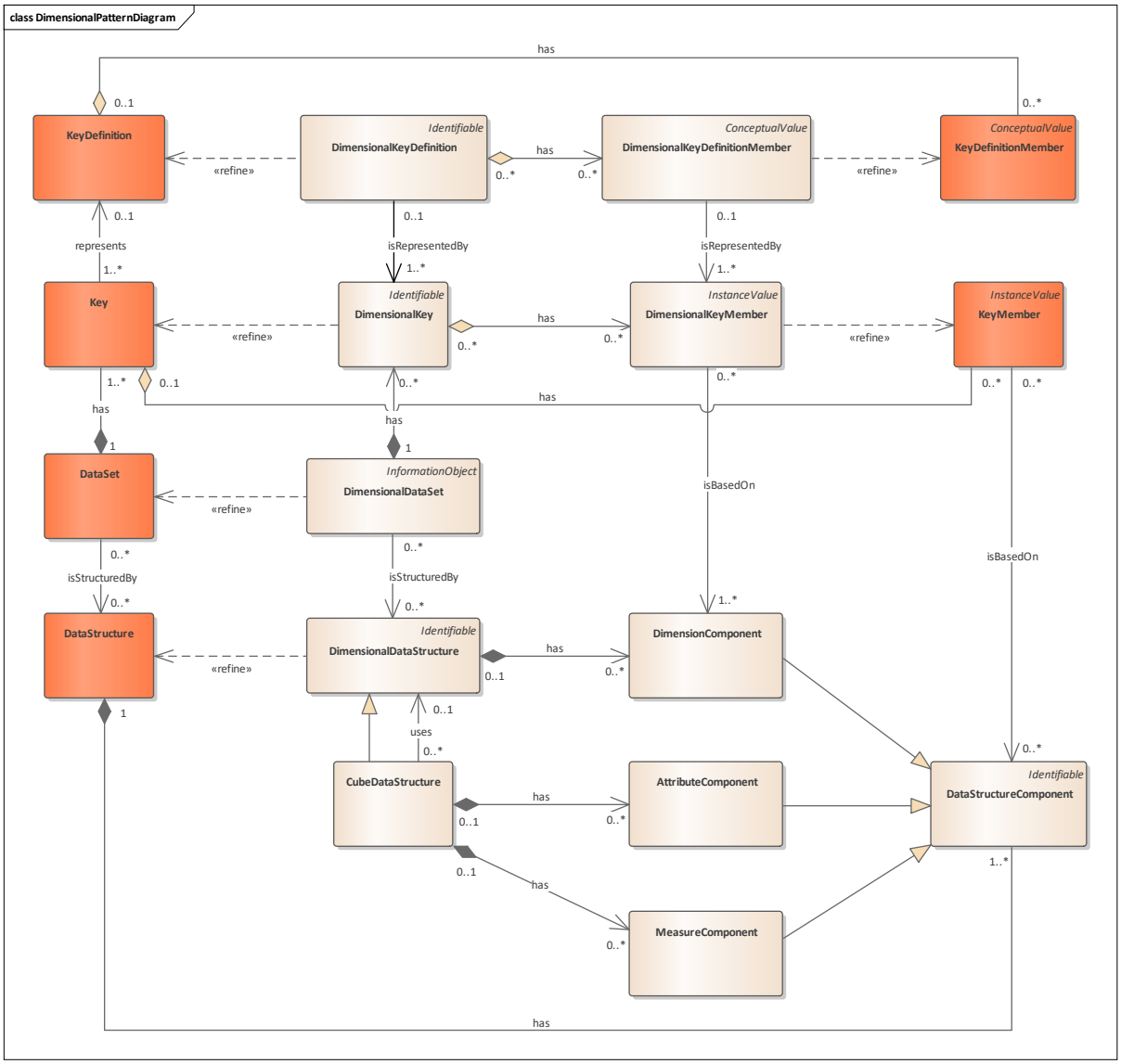


Figure 24: DimensionalPatternDiagram

The table below shows a long representation of a cube with three DimensionalComponents, one MeasureComponent, and two AttributeComponents. The attributes in this case indicate revised data in the cells of the cube, identified by vintage, and with an indication of what revision process took place.

DimensionalComponents *QualifiedMeasure* *AttributeComponents*

RefArea	Sex	TimePeriod	AverageLongevity	Vintage	Vintage Reason	Revision process
Newport	Female	2004 - 2006	80.7	**Aug-09**	additional data	recalculate mean
Newport	Female	2005 - 2007	80.9	**Aug-09**	additional data	recalculate mean
Newport	Female	2006 - 2007	81.5	**Aug-09**	no change	none
Newport	Male	2004 - 2006	76.7	**Aug-09**	additional data	recalculate mean
Newport	Male	2005 - 2007	77.1	**Aug-09**	additional data	recalculate mean
Newport	Male	2006 - 2007	77.0	**Aug-09**	additional data	recalculate mean
Cardiff	Male	2004 - 2006	78,7	**Aug-09**	additional data	recalculate mean
Etc.						

Figure 25: Attribute components for a cube – long representation

In the diagram below, we can see that multiple Datums can exist for those cases where there are revisions: these would share a Key but would be distinguished by the vintage property associated with each RevisableDatum.

While such revisions can be handled in other ways using this model (a time stamp associated with the observation – observation period - functioning as a dimension, for example) many systems use the approach modeled here, and do not manage revisions as part of the dimensionality of their data. The requirement is that two values with identical Keys be distinguishable – this model includes RevisableDatum to support those systems which require it.

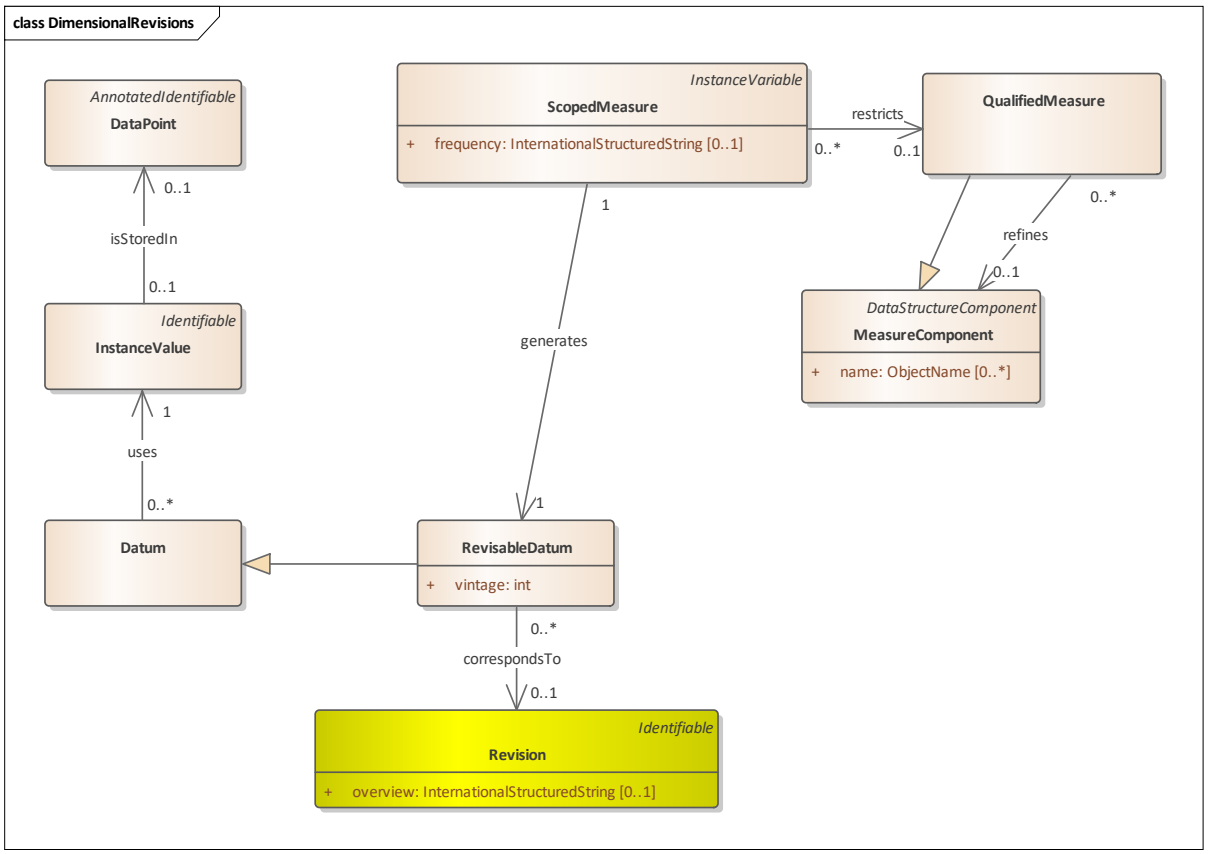


Figure 26: DimensionalRevisions

H. Key-Value Format

1. Example

A Key-Value store represents a repository holding data as a set of pairs, a key – the InstanceKey - and its associated value, a DataPoint. The DDI-CDI model is shown in figure 14. A key is a unique value that allows look-up of its linked value. The DDI-CDI model includes a KeyValueDataStore which contains the key-value pairs.

There are many possible ways to compose keys. The KeyValueDataStore may be divided into contexts, within which all of the subordinate keys are unique. The subject of the data – either a Unit or Population – can be contained as a component of the key. When this is a population, this portion of the key may itself be composed of the dimensional identifiers of the population, as for multi-dimensional data. Time may serve as a component of the key. Reference values may be used, as may variables. If needed, a "synthetic" component may be used, which holds no meaning but is unique within the context of the key.

In the example below the data are stored as key-value pairs. The Key column contains InstanceKey values that identify the associated DataPoints. Looking at the data in Figure 27, the value "3.3.1992" could be associated with a key "Marie-Born" combining the unit identifier ("Marie") and the variable

name (“Born”). The date 3.3.1932, for example is described by the InstanceKey “Marie-Born”. The cell containing 3.3.1932 is the DataPoint identified by the Key. This table, if combined with other data with keys composed in different ways, add a conxtext – a Contextual component – to the key to distinguish between the different ways in which data are being composed within the repository.

The KeyValue structure can be used for data in data lakes, No SQL systems, and other forms of big data.

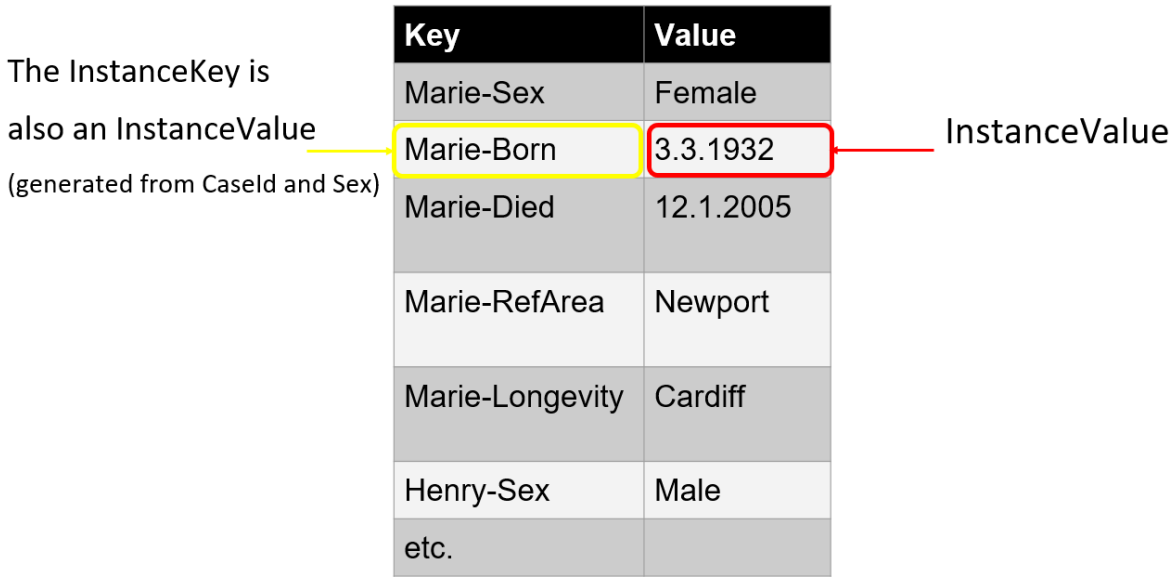


Figure 27: Key-Value Store

2. Discussion of structure and diagrams – Key-Value

At its heart the Key-Value model is simple. A key identifies a value, and a set of these are help in a KeyValueDataStore. The key is represented in DDI – CDI as an InstanceKey, the value as a DataPoint. The structure of the KeyValueDataStore is known from the KeyValueStructure with which it is associated.

It is possible to have more than one scheme for the composition of keys, by including in each a component which represents that scheme – or “context” – within which the key is unique.

The diagram below gives an overview of the relevant classes in DDI – CDI:

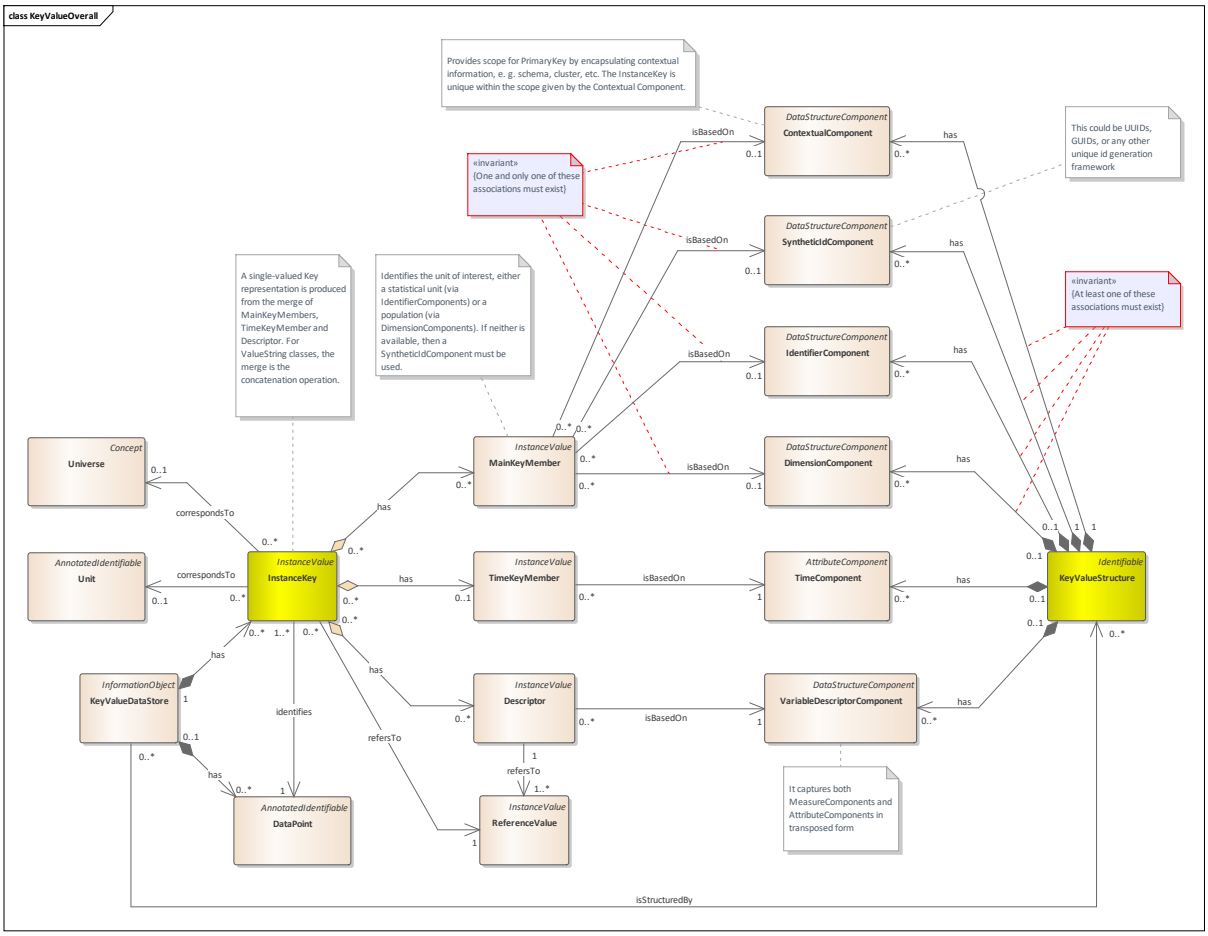


Figure 28: KeyValue overall diagram

InstanceKeys may be composed of a variety of different members: MainKeyMember, TimeKeyMember, and Descriptor are all used. These members are in turn composed of different StructureComponents according to rules which guarantee their uniqueness.

The members which are used to compose an InstanceKey are shiown in the diagram below:

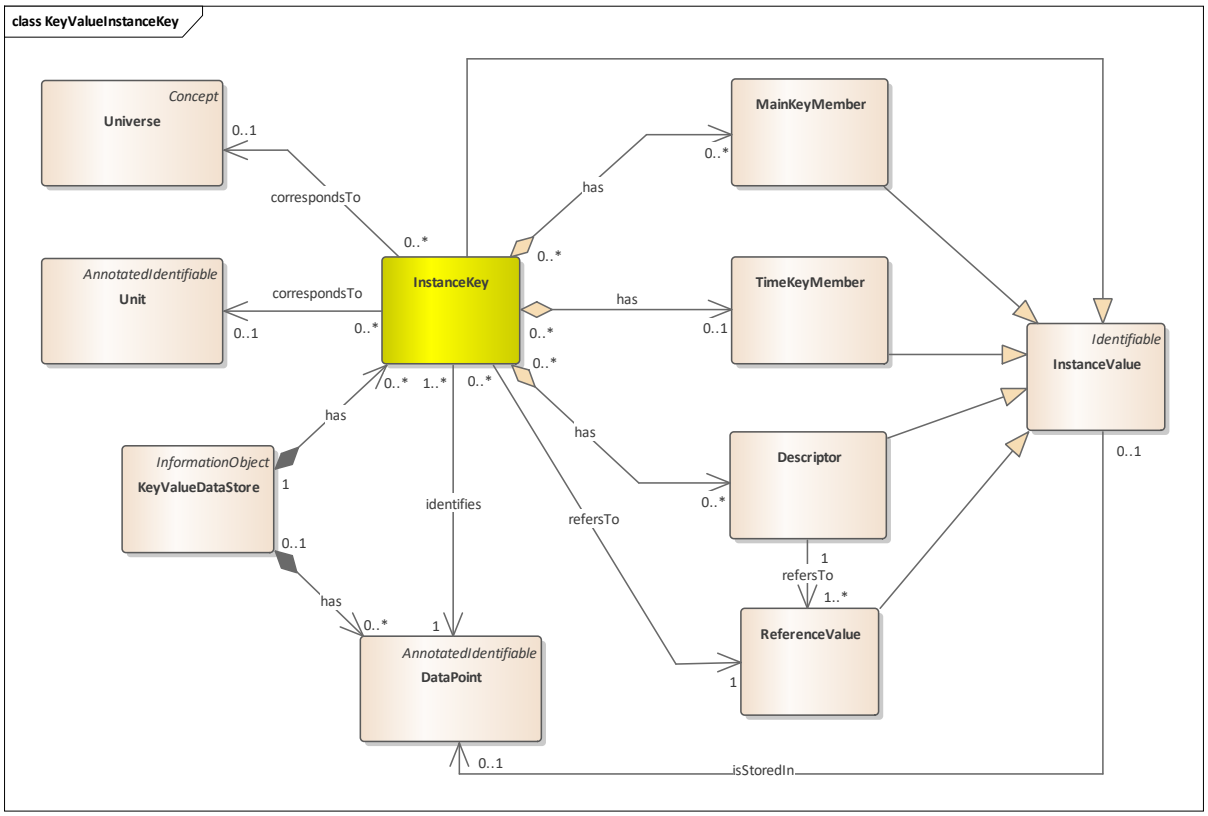


Figure 29: KeyValueInstanceKey

The MainKeyMember is the most complex one. In the simplest case, it may be composed of a SyntheticIdComponent, which might be a GUID or similar identifier which is guaranteed to be unique, but serves no other purpose. IdentifierComponents may be used to provide unitary values which identify the Units of the value (that is, their subject). Similarly, Units and Populations may be identified using DimensionComponents, providing a compound key structure like that found for multi-dimensional data. If more than one approach to composing keys is used, each may be established as a "context", and this can be added to the keys using the ContextualComponent.

TimeKeyMembers are made up of TimeComponents, which may be anything with a temporal association (this can be an enumerated value such as "Valid", a timestamp, or any other time-related value.)

Descriptors use the VariableDescriptorComponent, which brings together AttributeComponents and MeasureComponents (as for the Long Data structure). Descriptors are associated with a ReferenceValue – that is, the value held as an instance of the component being used to compose the key. (In our example, the variable "Born" could be a column in a Wide table, or a value in a Long table in the VariableDescriptor column. For Key-Value data, it is used as a Member in composing the Key.)

The StructureComponents making up the various Members may be seen in the diagram below:

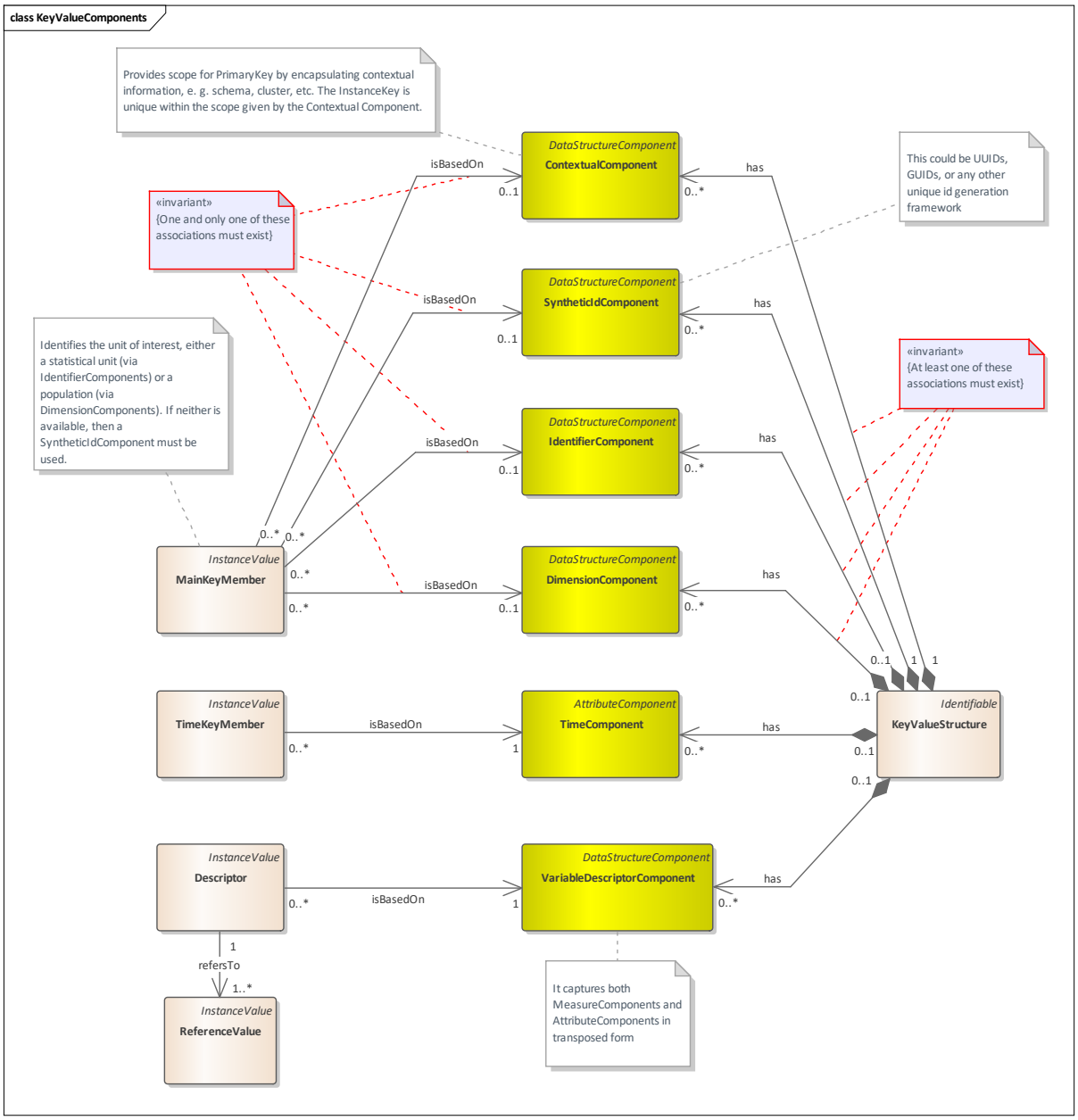


Figure 30: KeyValueComponents

A Key has a structure consisting of all of these components.

H. Physical Data Set (Wide Format)

The PhysicalDataSet diagram below shows the relationship of the PhysicalDataSet to other classes. A PhysicalDataset is a set of record segments (PhysicalRecordSegments). In older data files it was common to have a record (a row of a table) that was represented as a sequence of shorter records (e.g. strings) due to constraints imposed by the physical media. A record, for example, of 150 characters required two 80 column cards. A property of the PhysicalDataSet signifies the number of segments per record.

The order of the PhysicalRecordSegments is specified by a set of PhysicalRecordSegmentPositions, each of which having an integer describing the position of the segment in the dataset.

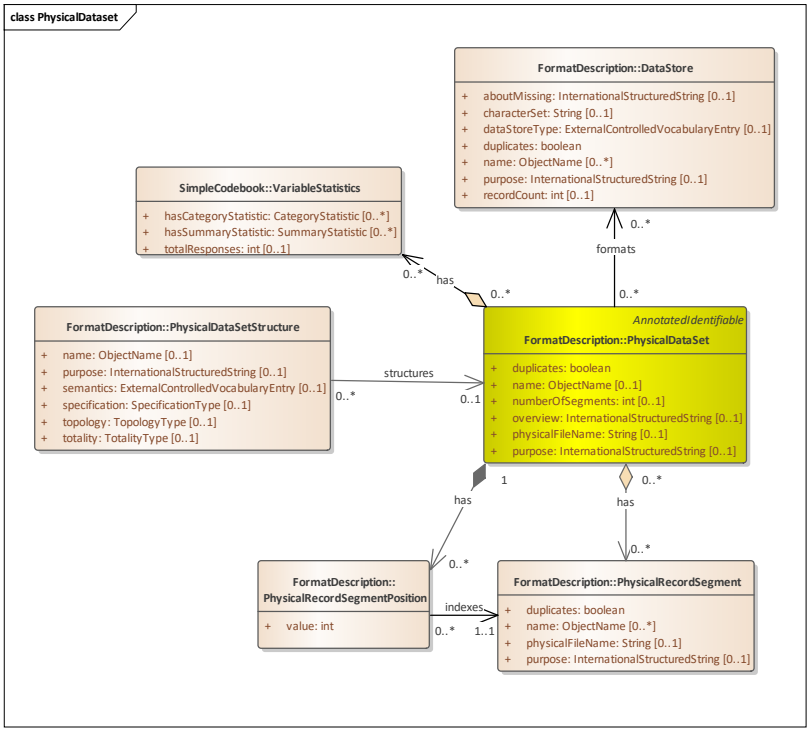


Figure 31: PhysicalDataSet overall diagram

The PhysicalRecordSegment is composed of DataPoints. A DataPoint contains an InstanceValue. In a text file the InstanceValue would be a substring of the string comprising the PhysicalRecordSegment. In a binary file it would be a sequence of bits within a larger sequence of bits. A DataPoint is described conceptually by an InstanceVariable. It is identified and set into context by a Key. The example below, for a traditional rectangular table, uses a WideKey.

The DataPoint is also described by a ValueMapping. For a string representation this contains information like the separator used for the decimal part of a number (defaultDecimalSeparator), or the maximum length of the string (maximumLength), etc.

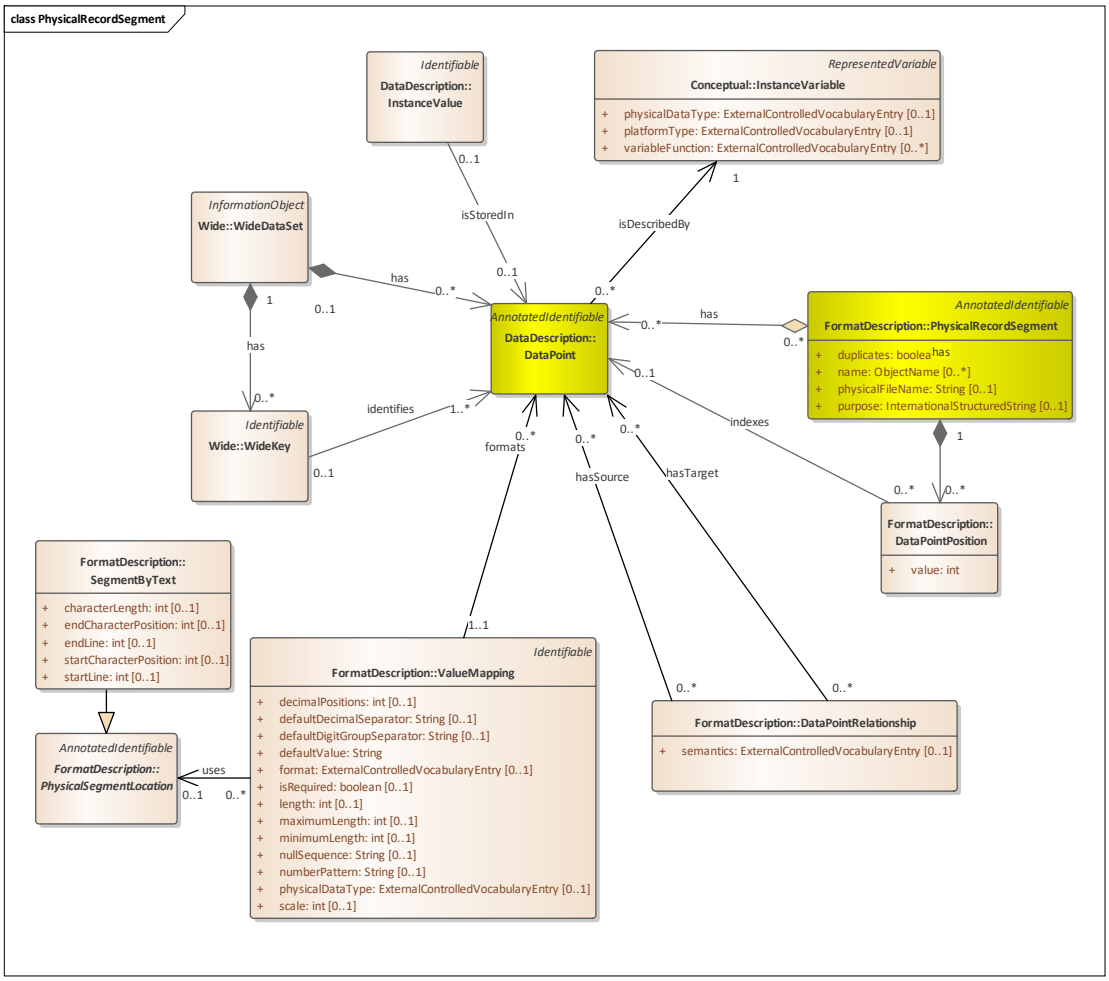


Figure 32: PhysicalRecordSegment diagram

In a text file the InstanceValue in a DataPoint is a substring of the PhysicalRecordSegment string itself. In a delimited file like a CSV file, the separation of those sequential substrings is indicated by delimiters. The PhysicalSegmentLayout contains information about those delimiters, the encoding of the record segment, whether text values are enclosed in quotes, etc..

For a fixed width file the ValueMapping can point to a SegmentByText object that contains information like the starting position (startCharacterPosition) and ending position (endCharacterPosition) of the substring within the segment. There is a parent class, PhysicalSegmentLocation, that will allow for description of data location in other types of media than text files. In a binary file this might be starting byte number and ending byte number. A video clip within a larger video file might be described by a start time and end time or by start and end frame number.

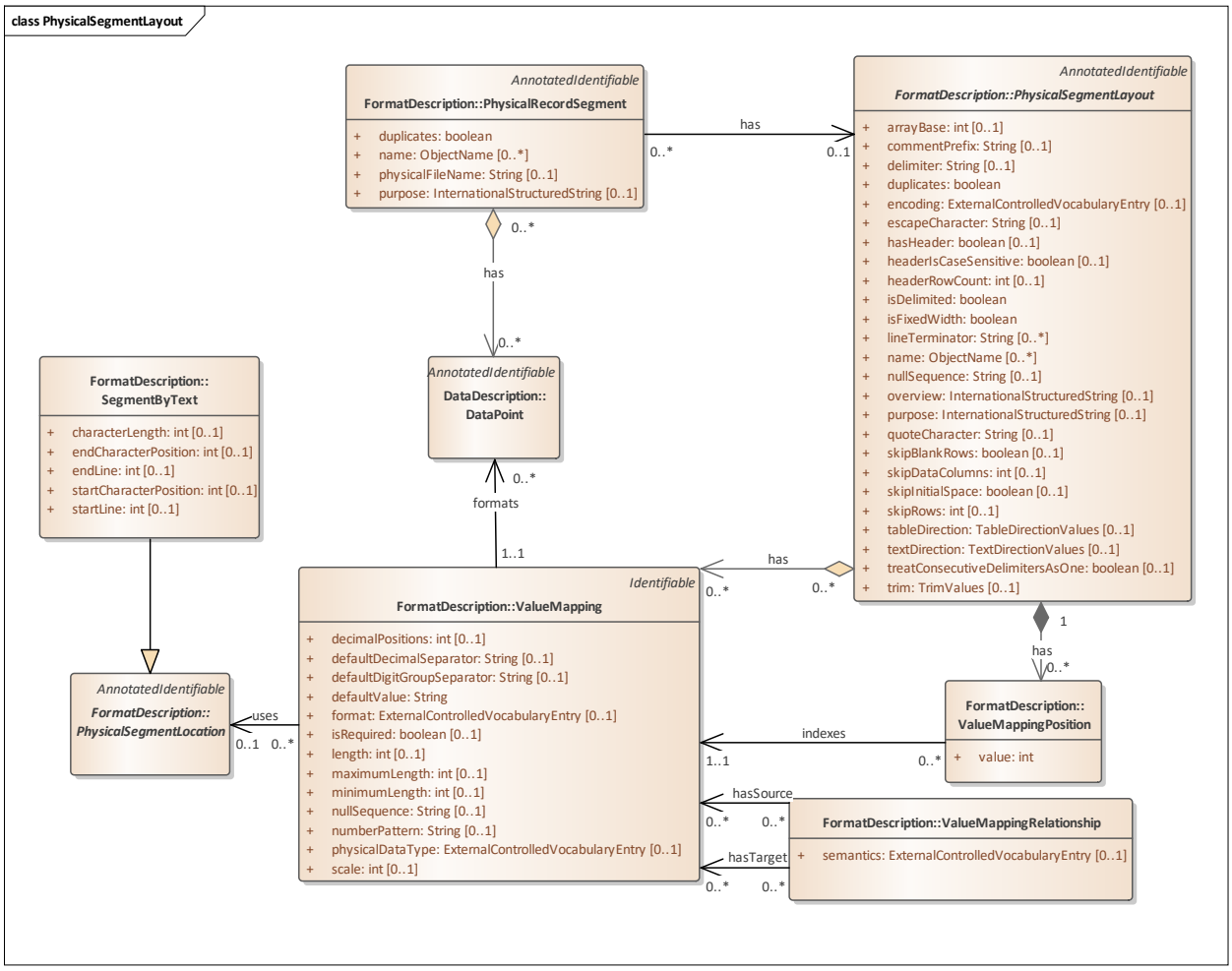


Figure 33: PhysicalSegmentLayout diagram

I. Transformations between Formats/Examples

1. Wide and Long: Correspondence between Unit record data and data in a Long format

Figure 34 below shows the mapping between the Wide Unit record format and the Long format. We see that all combinations of variables and values for each unit record identifier are retained. Each value in the record for Marie now has its own row, with a second value – the Unit Variable – telling us what the value is (the column in the Wide table). The cell value is the InstanceValue.

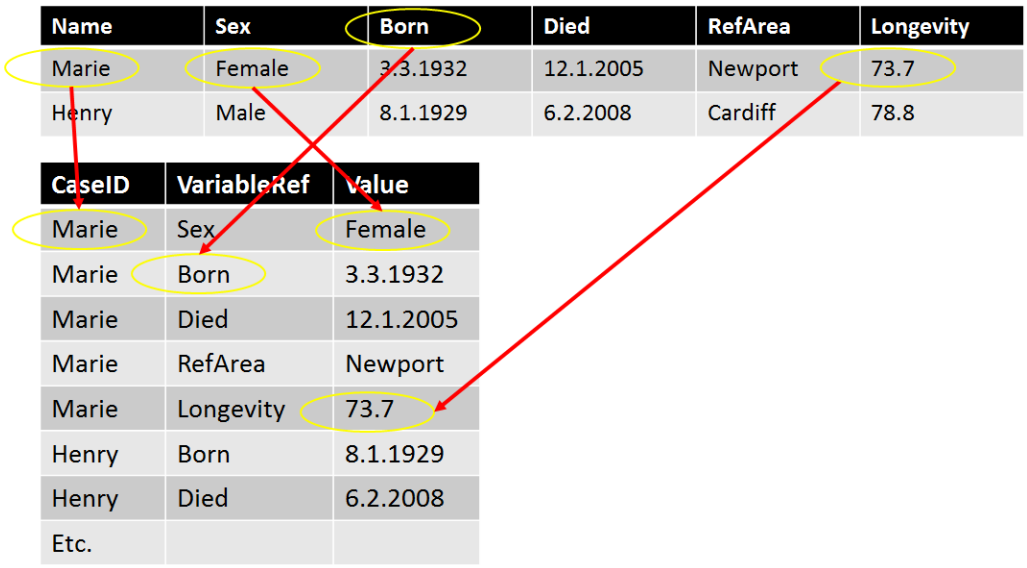


Figure 34: Transformations from Wide to Long format.

The VariableDescriptorComponent allows for the tracking of Datums between traditional wide layouts like the unit record format and Long layouts as shown in the Figure 34 example. All of the popular data analysis platforms have procedures like the R and Stata "reshape" function, or SAS PROC Transpose, that transform data tables back and forth between the two layouts. DDI – CDI provides a way to record this metadata which is not typically supported by non-proprietary formats.

Some types of data, like event data, typically employ Long layouts for the flexibility of adding measures and for the ability to represent sparse structures economically. Documenting these layouts with earlier versions of DDI (e.g., DDI Codebook, DDI Lifecycle) has been problematic. Columns like "Value" in the Long layout example cannot be described as a traditional variable with a single value domain. They are instead a set of Datums having different conceptual domains and representations.

The ValueMapping attached to the DataPoint allows for description of the physical representation of the generic representations in the Value column. That column as a whole must have a common representation, like a text string or bit string, that is capable of representing all of the value types for the set of underlying InstanceVariable.

2. Wide and dimensional: Unit record data tabulated into an aggregate data Cube

Unit record data can be tabulated into cubes (aggregate/dimensional data). Data from the individual units contribute to the aggregates of a cube. We see that 'Mary', 'Henry' and the others contribute to the aggregate statistics of the cube. The appropriate Unit record datum are averaged, producing the datum for the cube cell. In the cube below Marie contributes to two different cells due to overlapping time periods, while Henry only contributes to one cell.

PersonID	Sex	Born	Died	RefArea	Longevity
Marie	Female	3.3.1932	12.1.2005	Newport	73.7
Henry	Male	8.1.1929	6.2.2008	Cardiff	78.8
Etc.					

RefArea	Sex	TimePeriod	AverageLongevity
Newport	Male	2004-2006	76.7
Newport	Male	2005-2007	77.1
Newport	Male	2006-2008	77.0
Newport	Female	2004-2006	80.7
Newport	Female	2005-2007	80.9
Newport	Female	2006-2008	81.5
Cardiff	Male	2004-2006	78.7
Cardiff	Male	2005-2007	78.6
Cardiff	Male	2006-2008	78.7
Etc.			

Figure 35: Unit record data transformed to cube data

When computing a cube from the unit record data the value domains of some of the variables listed as measures above will correspond to dimensions of the cube. The categories of Sex, for example define the Sex dimension in the cube example. A computation on Died above would produce the time categories for the cube. The combination of dimension values for each unit (person here) would determine which set of units would contribute to the computation of the measure (Longevity here).

The SQL query that follows computes the cube data from the unit data:

```
create table WalesCube as
select Sex,RefArea,
    case
     when died >= '1jan2004'd and  died <= '31Dec2006'd   then "2004-2006"
     when died >= '1jan2005'd and  died <= '31Dec2007'd   then "2005-2007"
     when died >= '1jan2006'd and  died <= '31Dec2008'd   then "2006-2008"
    else " "
    end as TimePeriod,
    mean(Longevity) as Longevity

 from WalesUnitData
 group by sex, TimePeriod, RefArea
 ;
```

The processing code used to perform aggregations can be expressed in many different forms, both standard and proprietary. The Process model of DDI – CDI is designed to work with these to connect the metadata describing the data (both pre- and post-transformation) with the relevant processing code.

3. Long and Dimensional: Dimensional data represented in a Long data format

As noted before dimensional data can be represented in a Long layout. In this case the measure corresponds to the QualifiedMeasure in the model. Its population is the whole set of observations in the cube. There could be an extra column to represent the vintage instance for the associated measure. The DDI-CDI model includes classes that can assign roles to variables. In this example the first three variables take on the role as an IdentifierComponent. The values (codes), like “Newport”, or “2005-2007”in those columns are the representations of IdentifierComponent in the model. The longevity variable has a MeasureComponent, and the revision variable is an AttributeComponent. The values (codes) like “Newport”, or “2005-2007” in those columns are the representations of the IdentifierComponents in the model.

	IdentifierComponent		*QualifiedMeasure*	
Geography	**Gender**	**Time**	**Longevity**	**Vintage**
NewPort	Male	2004-2006	76.7	**Aug-09**
NewPort	Female	2004-2006	80.7	**Aug-09**
NewPort	Male	2005-2007	77.1	**Aug-09**
NewPort	Female	2005-2007	80.9	**Aug-09**
NewPort	Male	2006-2008	77	**Aug-09**
NewPort	Female	2006-2008	81.5	**Aug-09**
Cardiff	Male	2004-2006	78.7	**Aug-09**
Cardiff	Female	2004-2006	83.3	**Aug-09**
…	…	…	…	..
Merthyr	Male	2006-2008		**Jul-09**
Merthyr	Female	2006-2008		**Jul-09**

Figure 36: Dimensional as Long

4. Key-Value and Wide: Key-Value Stores in RAIRD

The example bellow shows what a possible dataset based on the RAIRD information model might look like. (RAIRD is a project involving the compilation of data from a set of administrative registers in Norway into a resource which can be used securely through an online analysis package by researchers. The central compiled data store is similar to the example given here, but researchers perform analysis on Wide data sets derived from it. The data is a form of “event history” data, giving information about specific events and periods for the Units it describes.)

RAIRD uses a hybrid form of Long and Wide layouts in that they add StartDate and EndDate as attributes that identify a value. In Figure 37 we recognize the crosswalk from the Wide Unit record data format to Long. StartDate and EndDate variables for each value are added additionally.

The keyValue table expresses the collection of variables in a possible RAIRD data set and how they are ordered. Key values link roles to each of them.

PersonID	Sex	Born	Died	RefArea	Longevity
Marie	Female	3.3.1932	12.1.2005	Newport	73.7
Henry	Male	8.1.1929	6.2.2008	Cardiff	78.8

raird:keyValue

InstanceVariableID	IndexValue	Role
CaseID	1	Identifier
VariableReference	2	Identifier
Value	3	Measure
StartDate	4	Attribute
EndDate	5	Attribute

RAIRD DataSet

CaseID	VariableRef	Value	StartDate	EndDate
Marie	Sex	Female	3.3.1932	12.1.2005
Marie	Born	3.3.1932	3.3.1932	12.1.2005
Marie	Died	12.1.2005	12.1.2005	12.1.2005
Marie	RefArea	Newport	1.1.2003	12.1.2005
Marie	Longevity	73.7	12.1.2005	12.1.2005
Henry	Born	8.1.1929	8.1.1929	
Henry	Died	6.2.2008	6.2.2008	
Etc.				

Figure 37: Example from the RAIRD information model

Here, we see that both CaseID and VariableRef function as identifiers – taken together, they uniquely identify a record in the Long format, and indeed as the identifier for a specific measure (the Value).

5. Time Series

With time as an attribute dimension in a full cube, a time series can be seen as a slice of the cube, holding the structural identifier values constant. In the example below geography and Gender are held constant and time varies across its possible values. The vintage column is added to indicate which revision of the data is being reported.

CellDefinition			*QualifiedMeasure*	
geography	**Gender**	**time**	**longevity**	**vintage**
NewPort	Male	2004-2006	76.7	**Aug-09**
NewPort	Male	2005-2007	77.1	**Aug-09**
NewPort	Male	2006-2008	77	**Aug-09**

6. Key-Value Stores and Streams

Streaming data may involve a flexible set of measures arriving at unpredictable times. Structures that may be useful for streaming data include the tall structure (like for event data) or a key value store. With a tall structure, measure variables may be associated with identifier variables (such as a sensor identifier) and attribute variables (such as time of measurement, time of receipt, and location of measurement).

Measures may involve datatypes note currently described in DDI (images, sound recording, etc.) but envisioned as potential candidates for inclusion in future.

An example sensor observation from the W3C Semantic Sensor Network Ontology (SSN) (https://www.w3.org/TR/vocab-ssn/#iphone_barometer-sosa) is of a barometric pressure taken by an iPhone. The SSN RDF for the Observation is:

```
<Observation/346344> rdf:type sosa:Observation ;
  sosa:observedProperty <sensor/35-207306-844818-
0/BMP282/atmosphericPressure> ;
  sosa:hasFeatureOfInterest  <earthAtmosphere> ;
  sosa:madeBySensor <sensor/35-207306-844818-0/BMP282> ;
  sosa:hasSimpleResult "1021.45 hPa"^^cdt:ucum ;
  sosa:resultTime "2017-06-06T12:36:12Z"^^xsd:dateTime .
```

A tall representation for the data might look like this, where the value atmosphericPressurehPa is a code that points to a variable that links to the Concept "`earthAtmosphere`" in units of hectoPascal (hPa).

SensorID	Property	Time	ResultingValue
sensor/35-207306-844818-0/BMP282	atmosphericPressurehPa	2017-06-06T12:36:12Z	1021.45

Figure 38: Sensor reading in Tall format

A Key-Value representation might look like this. The SensorID and Property are concatenated into a single Key. The Key could be decomposed into the SensorID and Property components as needed.

Key	Time	ResultingValue
sensor/35-207306-844818-0/BMP282/atmosphericPressure	2017-06-06T12:36:12Z	1021.45

Figure 39: Sensor reading in Key-Vaölue format

IV. The Process Model

A. Introduction

The D - CDI Process model is a generic process model able to describe retrospectively a succession of activities. These activities may be a set of business processes described at a conceptual level and/or a set of concrete steps (and their steps, ad infinitum) that take InformationObjects as parameters. Additionally, these activities may be a succession of questions in a questionnaire. InformationObjects may include, data, structured metadata, and computer programs.

Several forms of "succession" can be described. They fall into two categories – deterministic and non-deterministic. Deterministic succession may be parallel or sequential. Non-deterministic succession may be temporally ordered using algebras like Allen's interval algebra. Alternatively, non-deterministic succession may be governed by inference engines that form the basis for rule-based systems.

Generally speaking, each type of succession is supported by a set of control constructs. Together the control constructs form a plan or program that orchestrates a workflow. Depending on the control constructs, there are a myriad of workflow patterns.

1. Relation to other standards

There are several models currently in use which provide a strong basis for the DDI - CDI Process model. PROV-O is perhaps the best-known of these, giving us a basic set of classes describing *Activities* (the things which are done), *Agents* (the people and organizations which do things), and *Entities* (the resources which are operated on/with and produced). This is an extremely general model, and one which was designed to be made specific for use in specific applications.

Recently, PROV-O has been extended by ProvONE. ProvONE makes PROV-O data- and computer-program-specific. In PROV-O, entities didn't distinguish data at different level of specificityThe PROV-O Plan entity lacked the specificity to describe the structure of computer programs and the specific successions of activities (workflows) that programs create.

Here is the ProvONE Conceptual Model:

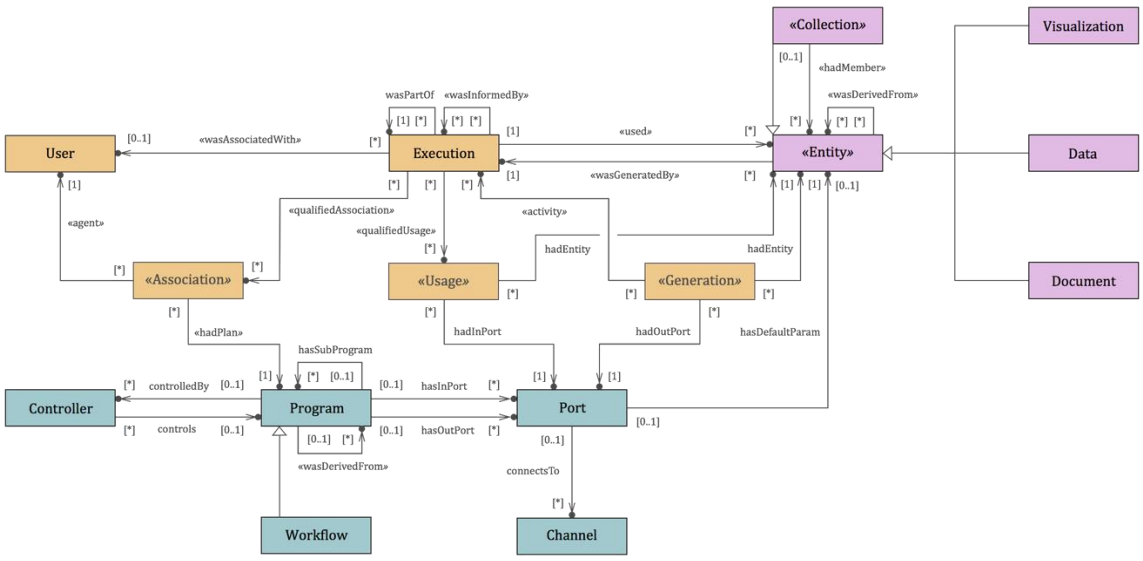


Figure 40: The PROVOne model

DDI - CDI process descriptions can be understood as extensions of PROV-O and ProvONE.

These extensions mostly take the form of ControlConstructs which DDI - CDI has borrowed from other products in the family of DDI specifications, notably DDI Lifecycle. DDI Lifecycle process components borrowed heavily from OWL-S, as shown below. (Notably, the Control Construct is a central feature of how DDI Lifecycle describes questionnaire flows.)

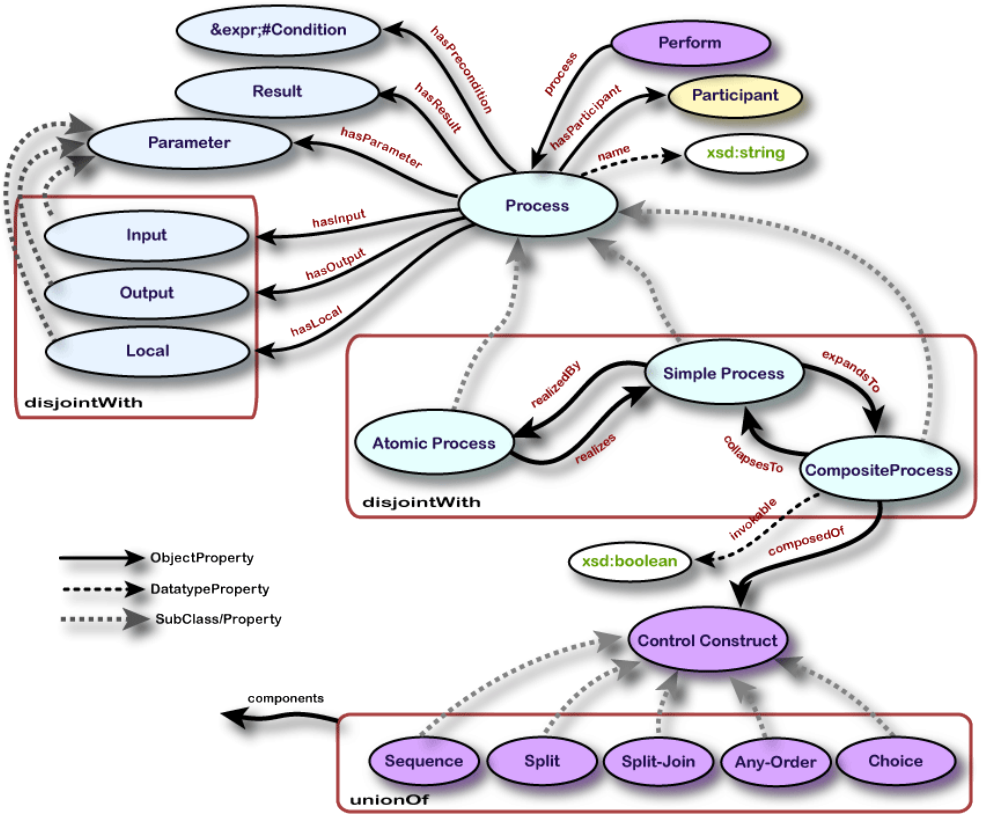


Figure 41: Control constructs mind map

2. Aspects covered by the DDI - CDI Process model

Currently “prospective provenance” and “process provenance” are not in scope. In prospective provenance plans and programs have a hand in guiding execution. Process provenance is about workflow evolution over time. Workflow evolution is integral to machine learning experiments which might evaluate a succession of workflows. (Workflow evolution may be addressed by DDI – CDI in future.)

For now, the focus is “retrospective provenance” or, again, “data lineage”. When data lineage enumerates a set of beginning and intermediate on-ramps in a workflow, it is *backward data lineage*. When data lineage enumerates a set of off ramps for InformationObjects that have entered the workflow upstream, this is *forward data lineage*. The DDI - CDI Process model aims to be able to describe both backward and forward data lineage.

B. Process Model Conceptual Model Overview

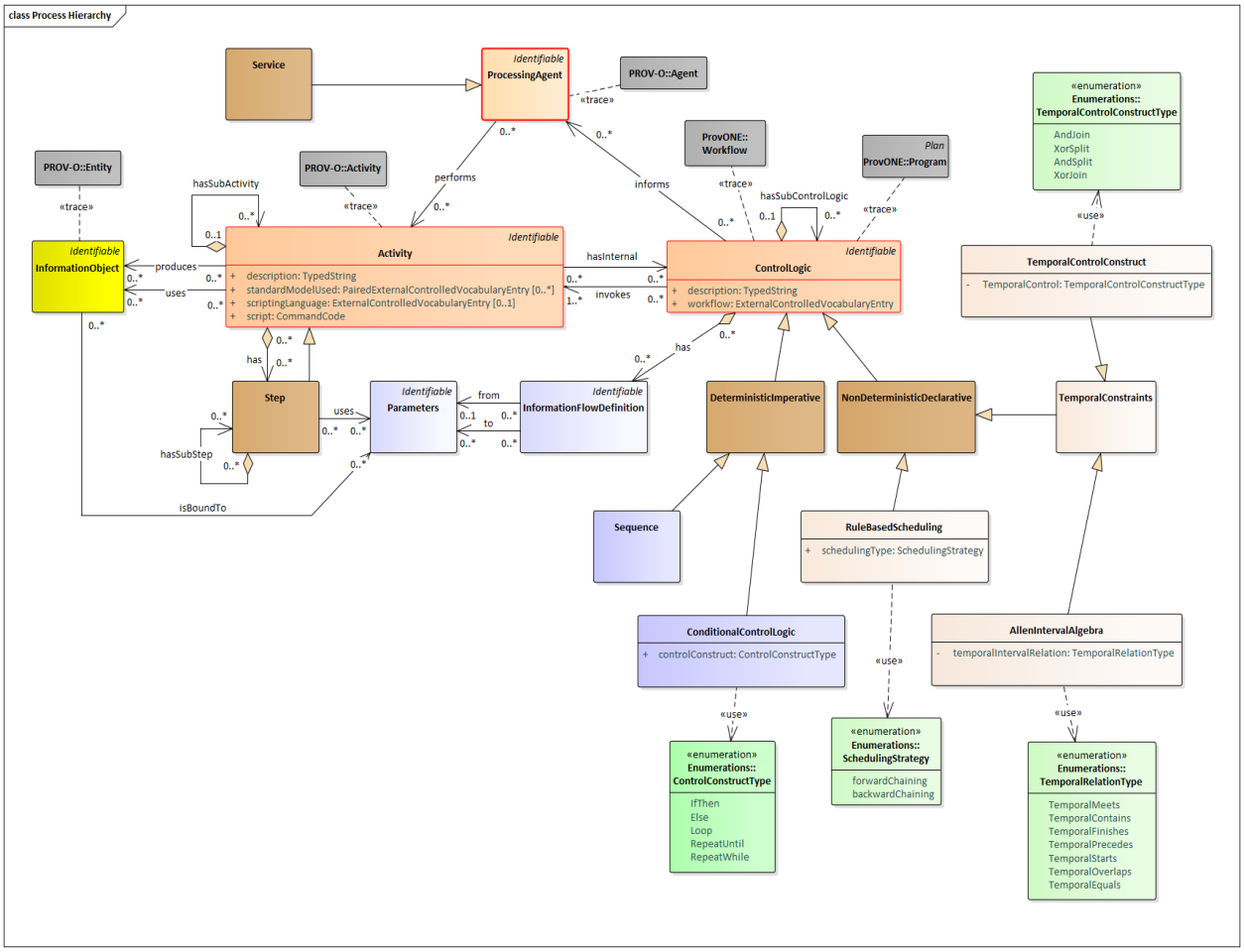


Figure 42: DDI - CDI Process model overview

In the DDI - CDI Process model ControlLogic invokes Activities. Activities may take In and Out Parameters. InformationObjects are bound to these Parameters. An InformationFlowDefinition connects the Out Parameter(s) of one Activity with the In Parameter(s) of another Activity. These connections, in the aggregate, create Workflows.

Note that an InformationObject may be data, structured metadata, or a program.

C. Process Model Conceptual Model Detail

Note that detailed documentation at for Data Description model in DDI – CDI can be found in this package in the folder: `\DDI-CDI Public Review 1\2 Model\Field-Level Doc\`.

1. ControlLogic

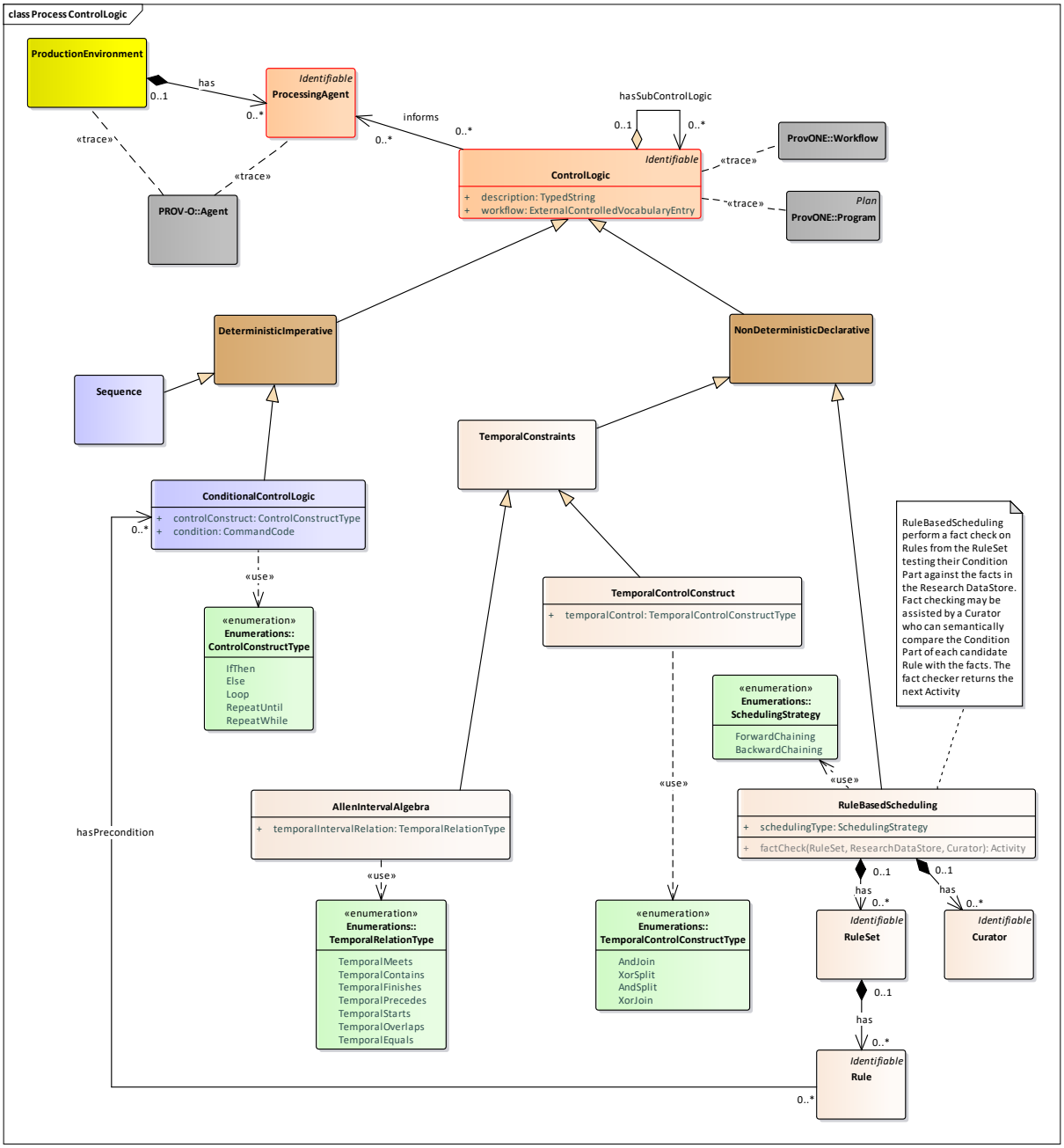


Figure 43: Process model control logic

Deterministic ControlLogic consists of Sequences and ConditionalControlLogic. Sequences may contain Sequences and ConditionalControlLogic. ConditionalControlLogic comes in several types or flavors including *If Then*, *Else*, etc. ConditionalControlLogic also includes logical expressions that evaluate to *true* or *false*. Finally, ConditionalControlLogic may contain Sequences.

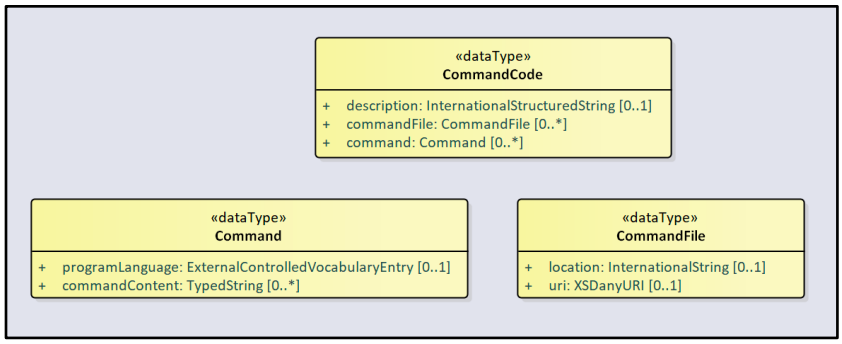
NonDeterministic ControlLogic has two subtypes – TemporalConstraints and RuleBasedScheduling. TemporalConstraints in turn has two subtypes – AllenIntervalAlgebra and TemporalControlConstruct. Both AllenIntervalAlgebra and TemporalControlConstruct use enumerations to qualify their type further. Note that AllenIntervalAlgebra is a calculus for temporal reasoning useful in describing complex pairwise temporal relationships across a group of Activities. TemporalControlConstructs, on the other hand, are useful in describing parallel processing.

RuleBasedScheduling takes a RuleSet and InformationObjects as input and produces InformationObjects as output. RuleBasedScheduling may employ the assistance of one or more domain-specific curators to match the Rule's conditions with real world facts or the current state of InformationObjects in the ResearchDataStore.

2. C2Metadata Support

DDI - CDI supports the work of the C2Metadata project. Both Activities and the Steps an Activity might contain in the Process Model host a Script made up of Commands. Each Command consists of programming language (specified in a codelist) and the Command content. Each Command may have multiple programming language / Command content pairs, which would be deemed equivalent (i.e., an executable STATA syntax example and its human-readable equivalent in SDTL – see the C2Metadata link above) .

C2Metadata takes a Script and its Command as input and produces additional Command content as output. The input is the original programming language and its command content. The output is documentation of the input. C2Metadata output may be in several languages including SDTL – the Structured Data Transformation Language.



Alongside SDTL, C2Metadata produces documentation of Commands in natural language and DDI Lifecycle.

3. Workflow

ControlLogic (the "program") in the DDI Process specification and all the deterministic and non-deterministic logic that inherit from ControlLogic have a *workflow* attribute. *workflow* is typed as ExternalControlledVocabulary (a codelist). In fact, the Workflow Patterns Initiative (WPI) has created a compendium of 40+ WorkflowPatterns noting the motivation, context, issues, issue solutions and supporting products and platforms associated with each one. In the Examples Document there are many examples in which the ExternalControlledVocabulary is the WPI where the WorkflowPattern in play in the ControlLogic is one of the 40+ patterns that the WPI describes.

```
<TemporalControlConstruct>
    <Agency>INDEPTH-Karonga-HDSS</Agency>
    <Id>Act-IntegrateData-1-TCC-AndSplit-1</Id>
    <Version>Pentaho-01</Version>
    <Workflow>
        <ControlledVocabularyAgencyName>Workflow Patterns Initiative</ControlledVocabularyAgencyName>
        <ControlledVocabularyID>WPI_Pattern_02</ControlledVocabularyID>
        <ControlledVocabularyName>Parallel Split</ControlledVocabularyName>
        <Content>
            The divergence of a branch into two or more parallel branches each of which execute concurrently.
        </Content>
        <Uri>http://www.workflowpatterns.com/patterns/control/basic/wcp2.php</Uri>
    </Workflow>
    <TemporalControl>AndSplit</TemporalControl>
</TemporalControlConstruct>
```

Several example WorkflowPatterns taken from the Workflow Patterns Initiative can be found below.

D. Illustrative WDI Workflow Patterns

Pattern 1 (Sequence)

FLASH animation of Sequence pattern

Description

A task in a process in enabled after the completion of a preceding task in the same process.

Synonyms

Sequential routing, serial routing.

Examples

The *verify-account* task executes after the credit card details have been captured.

The *codacil-signature* task follows the *contract-signature* task.

A receipt is printed after the train ticket is issued.

Motivation

The *Sequence* pattern serves as the fundamental building block for processes. It is used to construct a series of consecutive tasks which execute in turn one after the other. Two tasks form part of a *Sequence* if there is a control-flow edge from one of them to the next which has no guards or conditions associated with it.

Overview

Figure 1 illustrates the *Sequence* pattern using CP-nets.

Figure 1: Sequence pattern

Context

There is one context condition associated with this pattern: an instance of the *Sequence* pattern cannot be started again until it has completed execution of the preceding thread of control (i.e. all places such as p1 in the *Sequence* must be safe).

Here is the flash animation of that pattern.

Pattern 5 (Simple Merge)

FLASH animation of Simple Merge pattern

Description

The convergence of two or more branches into a single subsequent branch such that each enablement of an incoming branch results in the thread of control being passed to the subsequent branch.

Synonyms

XOR-join, exclusive OR-join, asynchronous join, merge.

Examples

At the conclusion of either the *bobcat-excavation* or the *D9-excavation* tasks, an estimate of the amount of earth moved is made for billing purposes.

After the *case-payment* or *provide-credit* tasks, initiate the *product-receipt* task.

Motivation

The *Simple Merge* pattern provides a means of merging two or more distinct branches without synchronizing them. As such, this presents the opportunity to simplify a process model by removing the need to explicitly replicate a sequence of tasks that is common to two or more branches. Instead, these branches can be joined with a simple merge construct and the common set of tasks need only to be depicted once in the process model.

Overview

Figure 5 illustrates the behaviour of this pattern. Immediately after either task A or B is completed, task C will be enabled. There is no consideration of synchronization.

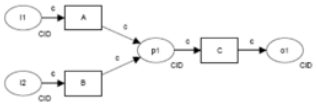


Figure 5: Simple merge pattern

Context

There is one context condition associated with the pattern: the place at which the merge occurs (i.e. place p1 fin Figure 5) is safe and can never contain more than one token.

Here is the flash animation of that pattern.

Pattern 40 (Interleaved Routing)

FLASH animation of Interleaved Routing pattern

Description

Each member of a set of tasks must be executed once. They can be executed in any order but no two tasks can be executed at the same time (i.e. no two tasks can be active for the same process instance at the same time). Once all of the tasks have completed, the next task in the process can be initiated.

Examples

The *check-oil, test-feeder, examine-main-unit* and *review-warranty* tasks all need to be undertaken as part of the machine-service process. Only one of them can be undertaken at a time, however they can be executed in any order.

Motivation

The *Interleaved Routing* pattern relaxes the partial ordering constraint that exists with the *Interleaved Parallel Routing* pattern and allows a sequence of tasks to be executed in any order.

Overview

Figure 58 illustrates the operation of this pattern. After A is completed, tasks B, C, D and E can be completed in any order. The **mutex** place ensures that only one of them can be executed at any time. After all of them have been completed, task F can be undertaken.

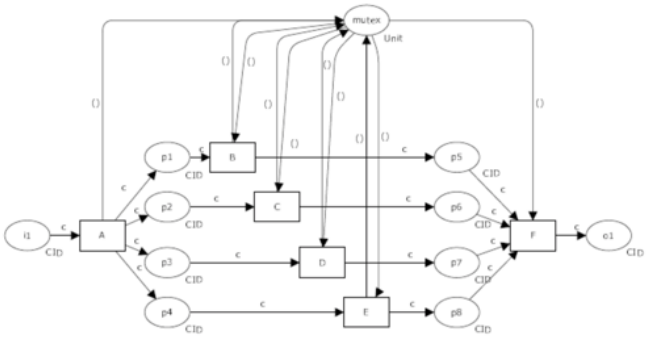


Figure 58: Interleaved routing pattern

Context

There is one consideration associated with the use of this pattern: tasks must be initiated and completed on a sequential basis, in particular it is not possible to suspend one task during its execution to work on another.

Here is the flash animation of that pattern.

A record of activities, agents and entities can be recorded in a Pathway, which provides the detailed provenance information for the data/metadata resulting from the process.

The diagram below provides a detail look at this portion of the DDI 4 State Based Process Model.

Pattern 3 (Synchronization)

FLASH animation of Synchronization pattern

Description

The convergence of two or more branches into a single subsequent branch such that the thread of control is passed to the subsequent branch when all input branches have been enabled.

Synonyms

AND-join, rendezvous, synchronizer.

Examples

The *despatch-goods* task runs immediately after both the *check-invoice* and *produce-invoice* tasks are completed.

Cash-drawer reconciliation can only occur when the store has been closed and the credit card summary has been printed.

Motivation

Synchronization provides a means of reconverging the execution threads of two or more parallel branches. In general, these branches are created using the *Parallel Split* (AND-split) construct earlier in the process model. The thread of control is passed to the task immediately following the synchronizer once all of the incoming branches have completed.

Overview

The behaviour of the *Synchronization* pattern is illustrated by the CP-net model in Figure 3. The pattern contains an implicit AND-join, known as the *synchronizer*, which is considered to be *activated* once it receives input on one of the incoming branches (i.e. at places p1 or p2). Similarly it is considered to be *reset* (and hence can be re-enabled) once input has been received on each incoming branch and the synchronizer has fired, removing these tokens.

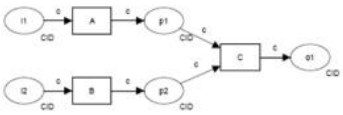


Figure 3: Synchronization pattern

Context

This pattern has the following context condition: once the *synchronizer* has been activated and has not yet been reset, it is not possible for another signal to be received on the activated branch or for multiple signals to be received on any incoming branch. In other words, all input places to the synchronizer (e.g. p1 and p2) are safe.

Here is the flash animation of that pattern.